计算机基础与实训教材系列

JSP动态网站开发实用教程

林巧民 主 编
肖艳 郑少京 陈远清 副主编



（理论→实例→上机→习题）4阶段教学模式

任务驱动的讲解方式，方便学习和教学

众多典型的实例操作，注重培养动手能力

PPT电子教案及素材免费下载，专业的网上技术支持

清华大学出版社

JSP 动态网站开发实用教程

林巧民 主编

肖艳 郑少京 陈远清 副主编

清华大学出版社

北 京

内 容 简 介

本书由浅入深、循序渐进地介绍了JSP的基础知识和相关技术。全书共分15章，分别介绍了JSP技术的概况及其基本工作原理，JSP运行和开发环境，JSP基本语法，Java语言编程技术，JSP内置对象，JSP + JavaBean的组合，Java Servlet技术，自定义JSP标记，JSP的安全性，数据库技术基础，在JSP中使用JDBC来访问数据库，可扩展标记语言XML，JSP应用的部署和错误处理等。最后一章还安排了JSP网站的构建实例，用于提高和拓宽读者对JSP的掌握与应用。

本书内容丰富，结构清晰，语言简练，图文并茂，具有很强的实用性和可操作性，是一本适合于大中专院校、职业院校及各类社会培训学校的优秀教材，也是广大初、中级电脑用户的自学参考书。

本书对应的电子教案、实例源文件和习题答案可以到 http://www.tupwk.com.cn/edu 网站下载。

图书在版编目(CIP)数据

JSP动态网站开发实用教程/林巧民 主编. —北京：清华大学出版社，2009.5

(计算机基础与实训教材系列)

ISBN 978-7-302-19836-9

Ⅰ.J…　Ⅱ. 林…　Ⅲ. JAVA语言—主页制作—程序设计—教材　Ⅳ.TP393.092

中国版本图书馆CIP数据核字(2009)第046748号

责任编辑：胡辰浩(huchenhao@263.net)　袁建华
装帧设计：孔祥丰
责任校对：成凤进
责任印制：
出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　编**：100084
社 总 机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：190×260　**印　张**：21.75　**字　数**：585千字
版　次：2009年5月第1版　**印　次**：2009年5月第1次印刷
印　数：1~5000
定　价：32.00元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。
联系电话：010-62770177转3103　产品编号：

丛书序

计算机已经广泛应用于现代社会的各个领域，熟练使用计算机已经成为人们必备的技能之一。因此，如何快速地掌握计算机知识和使用技术，并应用于现实生活和实际工作中，已成为新世纪人才迫切需要解决的问题。

为适应这种需求，各类高等院校、高职高专、中职中专、培训学校都开设了计算机专业的课程，同时也将非计算机专业学生的计算机知识和技能教育纳入教学计划，并陆续出台了相应的教学大纲。基于以上因素，清华大学出版社组织一线教学精英编写了这套“计算机基础与实训教材系列”丛书，以满足大中专院校、职业院校及各类社会培训学校的教学需要。

一、丛书书目

本套教材涵盖了计算机各个应用领域，包括计算机硬件知识、操作系统、数据库、编程语言、文字录入和排版、办公软件、计算机网络、图形图像、三维动画、网页制作以及多媒体制作等。众多的图书品种可以满足各类院校相关课程设置的需要。

◉ 已出版的图书书目

《计算机基础实用教程》	《中文版 Excel 2003 电子表格实用教程》
《计算机组装与维护实用教程》	《中文版 Access 2003 数据库应用实用教程》
《五笔打字与文档处理实用教程》	《中文版 Project 2003 实用教程》
《电脑办公自动化实用教程》	《中文版 Office 2003 实用教程》
《中文版 Photoshop CS3 图像处理实用教程》	《JSP 动态网站开发实用教程》
《Authorware 7 多媒体制作实用教程》	《Mastercam X3 实用教程》
《中文版 AutoCAD 2009 实用教程》	《Director 11 多媒体开发实用教程》
《AutoCAD 机械制图实用教程(2009 版)》	《中文版 Indesign CS3 实用教程》
《中文版 Flash CS3 动画制作实用教程》	《中文版 CorelDRAW X3 平面设计实用教程》
《中文版 Dreamweaver CS3 网页制作实用教程》	《中文版 Windows Vista 实用教程》
《中文版 3ds Max 9 三维动画创作实用教程》	《电脑入门实用教程》
《中文版 SQL Server 2005 数据库应用实用教程》	《中文版 3ds Max 2009 三维动画创作实用教程》
《中文版 Word 2003 文档处理实用教程》	《Excel 财务会计实战应用》
《中文版 PowerPoint 2003 幻灯片制作实用教程》	

⊙ 即将出版的图书书目

《Oracle Database 11g 实用教程》	《中文版 Pro/ENGINEER Wildfire 5.0 实用教程》
《中文版 Premiere Pro CS3 多媒体制作实用教程》	《ASP.NET 3.5 动态网站开发实用教程》
《Java 程序设计实用教程》	《中文版 Office 2007 实用教程》
《Visual C#程序设计实用教程》	《中文版 Word 2007 文档处理实用教程》
《网络组建与管理实用教程》	《中文版 Excel 2007 电子表格实用教程》
《AutoCAD 建筑制图实用教程（2009 版）》	《中文版 PowerPoint 2007 幻灯片制作实用教程》
《中文版 Photoshop CS4 图像处理实用教程》	《中文版 Access 2007 数据库应用实例教程》
《中文版 Illustrator CS4 平面设计实用教程》	《中文版 Project 2007 实用教程》
《中文版 Flash CS4 动画制作实用教程》	《中文版 CorelDRAW X4 平面设计实用教程》
《中文版 Dreamweaver CS4 网页制作实用教程》	《中文版 After Effects CS4 视频特效实用教程》
《中文版 Indesign CS4 实用教程》	《中文版 Premiere Pro CS4 多媒体制作实用教程》

二、丛书特色

1、选题新颖，策划周全——为计算机教学量身打造

本套丛书注重理论知识与实践操作的紧密结合，同时突出上机操作环节。丛书作者均为各大院校的教学专家和业界精英，他们熟悉教学内容的编排，深谙学生的需求和接受能力，并将这种教学理念充分融入本套教材的编写中。

本套丛书全面贯彻“理论→实例→上机→习题”4 阶段教学模式，在内容选择、结构安排上更加符合读者的认知习惯，从而达到老师易教、学生易学的目的。

2、教学结构科学合理，循序渐进——完全掌握“教学”与“自学”两种模式

本套丛书完全以大中专院校、职业院校及各类社会培训学校的教学需要为出发点，紧密结合学科的教学特点，由浅入深地安排章节内容，循序渐进地完成各种复杂知识的讲解，使学生能够一学就会、即学即用。

对教师而言，本套丛书根据实际教学情况安排好课时，提前组织好课前备课内容，使课堂教学过程更加条理化，同时方便学生学习，让学生在学习完后有例可学、有题可练；对自学者而言，可以按照本书的章节安排逐步学习。

3、内容丰富、学习目标明确——全面提升“知识”与“能力”

本套丛书内容丰富，信息量大，章节结构完全按照教学大纲的要求来安排，并细化了每一章内容，符合教学需要和计算机用户的学习习惯。在每章的开始，列出了学习目标和本章重点，

便于教师和学生提纲挈领地掌握本章知识点，每章的最后还附带有上机练习和习题两部分内容，教师可以参照上机练习，实时指导学生进行上机操作，使学生及时巩固所学的知识。自学者也可以按照上机练习内容进行自我训练，快速掌握相关知识。

4、实例精彩实用，讲解细致透彻——全方位解决实际遇到的问题

本套丛书精心安排了大量实例讲解，每个实例解决一个问题或是介绍一项技巧，以便读者在最短的时间内掌握计算机应用的操作方法，从而能够顺利解决实践工作中的问题。

范例讲解语言通俗易懂，通过添加大量的“提示”和“知识点”的方式突出重要知识点，以便加深读者对关键技术和理论知识的印象，使读者轻松领悟每一个范例的精髓所在，提高读者的思考能力和分析能力，同时也加强了读者的综合应用能力。

5、版式简洁大方，排版紧凑，标注清晰明确——打造一个轻松阅读的环境

本套丛书的版式简洁、大方，合理安排图与文字的占用空间，对于标题、正文、提示和知识点等都设计了醒目的字体符号，读者阅读起来会感到轻松愉快。

三、读者定位

本丛书为所有从事计算机教学的老师和自学人员而编写，是一套适合于大中专院校、职业院校及各类社会培训学校的优秀教材，也可作为计算机初、中级用户和计算机爱好者学习计算机知识的自学参考书。

四、周到体贴的售后服务

为了方便教学，本套丛书提供精心制作的 PowerPoint 教学课件(即电子教案)、素材、源文件、习题答案等相关内容，可在网站上免费下载，也可发送电子邮件至 wkservice@vip.163.com 索取。

此外，如果读者在使用本系列图书的过程中遇到疑惑或困难，可以在丛书支持网站(http://www.tupwk.com.cn/edu)的互动论坛上留言，本丛书的作者或技术编辑会及时提供相应的技术支持。咨询电话：010-62796045。

JSP 是由美国 Sun 公司倡导、许多公司参与建立的一种动态网页技术标准，它采用 Java 语言作为脚本语言，是 J2EE 体系结构中的一项重要组成技术，它为开发人员提供了一个 Server 端框架，基于这个框架，开发人员可以综合应用 HTML、XML、Java 语言以及其他脚本语言，灵活、快速地创建各种动态网页内容，是众多 Web 编程语言中的佼佼者。

本书从教学实际需求出发，合理安排知识结构，从零开始、由浅入深、循序渐进地讲解 JSP 的相关知识和开发技术，本书共分 15 章，主要内容如下：

第 1 章介绍了 JSP 技术的概况及其基本工作原理。

第 2 章介绍了 JSP 的运行和开发环境。

第 3 章介绍了 JSP 的基本语法，并解释其基本功能和作用。

第 4 章介绍了 Java 语言的编程基础。

第 5 章介绍了 Java 语言的面向对象编程技术。

第 6 章介绍了 JSP 的内置对象。

第 7 章介绍了 JSP＋JavaBean 的组合。

第 8 章介绍了 Java Servlet 技术。

第 9 章介绍了自定义 JSP 标记。

第 10 章介绍了 JSP 的安全性问题。

第 11 章介绍了数据库技术的基础知识。

第 12 章介绍了 JDBC 的相关知识以及如何在 JSP 中使用 JDBC 来访问数据库。

第 13 章介绍了可扩展标记语言 XML。

第 14 章介绍了 JSP 应用的部署和错误处理。

第 15 章介绍了 JSP 网站的构建实例。

本书图文并茂，条理清晰，通俗易懂，内容丰富，在讲解每个知识点时都配有相应的实例，方便读者上机实践。同时，在难于理解和掌握的部分内容上给出相关提示，让读者能够快速地提高操作技能。此外，本书还配有大量的综合实例和练习，让读者在不断的实际操作中更加牢固地掌握书中讲解的内容。

本书由林巧民主编，同时它也是集体智慧的结晶，参加本书编写和制作的人员还有肖艳、郑少京、陈远清、袁薇薇、陈晓静、陈映钳、赵臻、高俊、肖云龙、陈建兵、张兴武、周惠、杨玉敏、张凤霞、李志伟、张志云等人。由于作者水平有限，本书不足之处在所难免，欢迎广大读者批评指正。我们的邮箱是：huchenhao@263.net，电话：010-62796045。

作者
2009 年 3 月

推荐课时安排

章　　名	重 点 掌 握 内 容	教 学 课 时
第 1 章　初识 JSP	1. Web 结构 2. JSP 与 Java 和 Servlet 的关系 3. JSP 的运行原理	2 学时
第 2 章　JSP 运行环境和开发环境	1. JSP 的运行环境 2. Tomcat 的安装和配置 3. Eclipse 的安装 4. JSP 的开发方式	2 学时
第 3 章　JSP 语法	1. JSP 容器 2. JSP 的基本结构 3. JSP 的语法 4. JSP 指令 5. JSP 操作	4 学时
第 4 章 Java 编程语言	1. Java 的数据类型 2. 流程控制和跳转语句 3. 在 Eclipse 中开发 Java 程序	4 学时
第 5 章 Java 面向对象编程	1. 类和对象 2. 访问控制符 3. 继承与多态	4 学时
第 6 章 JSP 中的内置对象	1. JSP 的内置对象 2. out 对象 3. request 与 response 对象 4. session 对象 5. page 对象	4 学时
第 7 章 JSP 与 JavaBean	1. JavaBean 分类 2. JavaBean 的属性 3. JavaBean 持久化 4. JSP 上的 JavaBean	4 学时

(续表)

章　名	重点掌握内容	教学课时
第 8 章 Servlet 技术	1. Servlet 的优点 2. Servlet 的生命周期 3. JSP 与 Servlet 的区别	4 学时
第 9 章 JSP 标记库	1. 标记库描述符文件 2. 标记处理类 3. 自定义标记的生命周期 4. 定义脚本变量的标记	4 学时
第 10 章 JSP 安全性	1. 应用程序的安全性 2. Web 认证 3. Servlet 容器认证	2 学时
第 11 章 数据库基础	1. 关系数据库 2. SQL 语言 3. 数据库对象 4. SQL Server 的安装与使用	4 学时
第 12 章 JSP 数据库应用	1. 数据库驱动程序 2. JDBC 核心 API 3. 使用 JDBC 访问数据库 4. 数据库事务	2 学时
第 13 章 JSP 与 XML	1. XML 的语法结构 2. DTD 和 Schema 3. 使用 DOM 操作 XML 文件 4. 使用 SAX 操作 XML 文件	2 学时
第 14 章 JSP 应用的部署和错误处理	1. JSP 应用的部署 2. JSP Web 应用配置 3. JSP 错误处理	2 学时
第 15 章 JSP 网站的构建实例	1. 数据库设计与连接类 2. 核心 JavaBean 类 3. 性能测试	4 学时

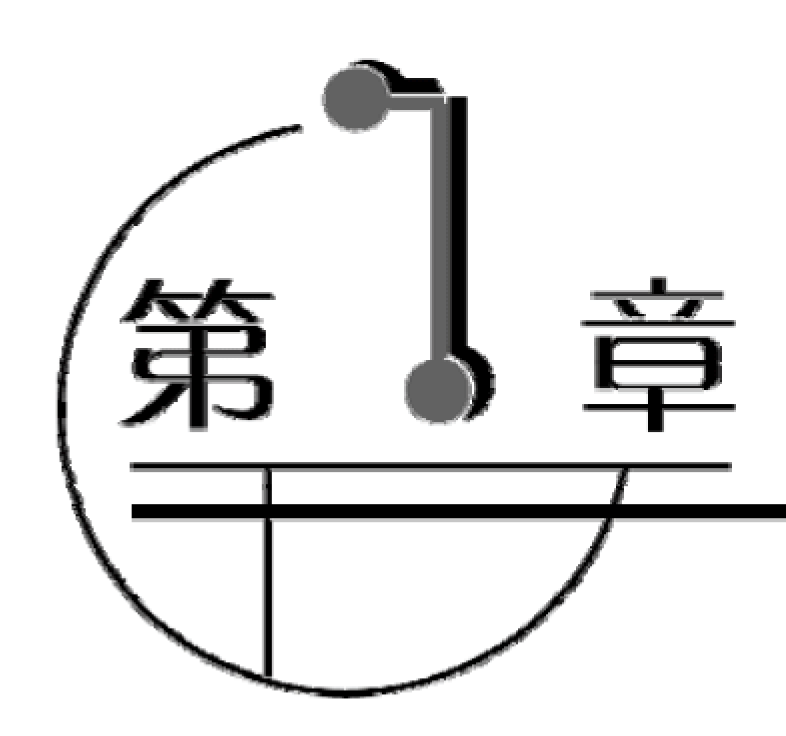
注：1、教学课时安排仅供参考，授课教师可根据情况作调整。

2、建议每章安排与教学课时相同时间的上机练习。



目 录

CONTENTS



初 识 JSP

学习目标

JSP 的全称为 Java Server Pages(中文译名为 Java 服务器页面)，是由 SUN (太阳微系统公司)主导的 Java 社群推出的用于 Web 应用服务的一种编程技术。JSP 技术采用在 HTML 中嵌入 Java 代码的方式，使开发人员能够比较轻松地编写出功能强大的动态网页。

本章将介绍 HTML 语言的基础知识，Web 服务器和应用服务器的概念，Java、Servlet 以及 JSP 技术的关系。本章的目标是让读者了解 JSP 技术的概况及其基本工作原理，并为后面的学习打下一定基础。

本章重点

- Web 结构
- JSP 与 Java 和 Servlet 的关系
- JSP 的运行原理

1.1 HTML 基础

在讲解 JSP 之前，有必要对编写静态网页的 HTML 语言做一下介绍，因为 JSP 页面的源代码中有很大一部份其实就是 HTML 语言，而其他 Java 代码块在经过应用服务器的解析后也将以 HTML 语言的形式提供给客户的浏览器进行解析显示。

1.1.1 概述

HTML(Hyper Text Markup Language)超文本标识语言，所谓超文本，是指除了文本内容外，

还可以表现图形、图像、音频、视频、链接等非文本要素。事实上，Internet 上众多网页的基础，就是 HTML 语言，特别是在互联网发展的初期，几乎所有的网页都是直接用 HTML 语言来书写的，随着后来应用需求的发展，要求让网页能够“动”起来，因此才在 HTML 页面中引入了脚本语言，像 JavaScript、VBScript、JScript 等脚本来作一些特殊控制，再到后来，应用的复杂性又要求引入更强大的编程语言，比如现在很多网站就采用各种各样的 Web 编程语言，如 JSP(Java Server Pages)、ASP(Active Server Pages)、PHP(Hypertext Pre-processor)等来进行 Web 应用的开发，这些 Web 编程语言可以实现诸如网络通信及数据库访问等功能，不过，这些复杂的 Web 网页在经过特定 Web 应用服务器的解析后，仍然是以 HTML 页面的形式输出的，因此，远程客户端在收到这些最终的 HTML 页面后，就可以用 IE 或者 FireFox 等浏览器来浏览网页内容了。总的来说，是 HTML 开启了互联网的 Web 应用，它是 Web 编程的最基础语言。

虽然 HTML 能够表现超文本格式的内容，但其文件本身仍旧是文本格式的。因此，需要用支持文本文件格式的编辑器来进行编辑，比如 Windows 提供的记事本和写字板就是最简单的 HTML 文件编辑器。若是使用诸如 Microsoft Word 等文档编辑器，在编辑 HTML 文件时也应使用其文本文件格式，而不应使用格式文档的功能，否则浏览器将无法正确解析。IDM 公司出品的 UltraEdit 软件，是一款功能全面、高效的专业文本编辑程序，对于专业的网页设计者来说，该软件是非常不错的选择，当然也可以选择另一类似软件 EditPlus。除此之外，其实更常用的网页编辑软件是 FrontPage、DreamWeaver 等可视化的工具，这些工具允许网页设计者一边设计，一边查看效果，因此一经推出，这些可视化网页编辑工具软件便迅速占领了相应市场，赢得了大多数用户。

HTML 描述网页文件内容的方法是通过引入一些功能符号(又叫标签，Tag)，来标记出各种特定效果，再由浏览器来解读 HTML 的这些功能符号，将文件内容的效果展示出来，HTML 是一种标记式的语言。在 HTML 网页里，图形图像、声音视频等必须利用其他软件来制作，再用 HTML 的标签将其标记在网页文件中某个位置，然后浏览器才能根据相应标签的功能对其进行解读，并在屏幕上显示相应效果。因此，学习 HTML，最主要的就是学习它的各种标签的功能，一般地，HTML 标签多数是成对出现的，另外，需要说明的是，HTML 语言是不区分大小写的！下面就具体介绍 HTML 的基本结构及其常用的一些标签功能。

1.1.2 基本结构

HTML 的基本结构是由文档头、文档体构成的，如下所示：

```
<HTML>
    <HEAD>
        头部信息
    </HEAD>
    <BODY>
        文档主体，正文部分
```



```
    </BODY>
</HTML>
```

下面是一个最基本的超文本文档的源代码：

```
<HTML>
  <HEAD>
    <TITLE>一个简单的 HTML 示例</TITLE>
  </HEAD>
  <BODY>
    <CENTER>
      <H3>欢迎光临我的主页</H3>
      <BR><HR>
 <FONT SIZE=2>这是我第一次做主页，无论怎么样，我都会努力做好！        </FONT>
    </CENTER>
  </BODY>
</HTML>
```



它的浏览效果如图 1-1 所示。



图 1-1　浏览效果

超文本中的标签可以分为：单标签(较少)和双标签(居多)。

单标签的形式：<标签名称>，如上述的
，它用来实现换行操作。

双标签的形式：<标签名称 属性 1 属性 2 属性 3 … > 内 容</ 标签名称>，如上述的字体 Font 标签(<FONT SIZE=2>这是我第一次做主页，无论怎么样，我都会努力做好！</FONT>)，“这是我第一次做主页，无论怎么样，我都会努力做好！”是内容，而 SIZE=2 是属性，属性可以有多个，比如 Font 标签中还可以有 color 颜色属性、style 风格属性等。

1.1.3 基本标签

1. 页面布局与文字设计

页面布局与文字设计主要涉及：标题、换行
、段落标签<P>、水平线段<HR>、文字的大小设置、文字的字体与样式、文字的颜色、位置控制等。

HTML 中提供了相应的标题标签<Hn>, 总共提供 6 个等级，n 越小，标题字号就越大。

<H1>…</H1>　　第一级标题
<H2>…</H2>　　第二级标题
<H3>…</H3>　　第三级标题
<H4>…</H4>　　第四级标题
<H5>…</H5>　　第五级标题
<H6>…</H6>　　第六级标题

其浏览效果如图 1-2 所示。

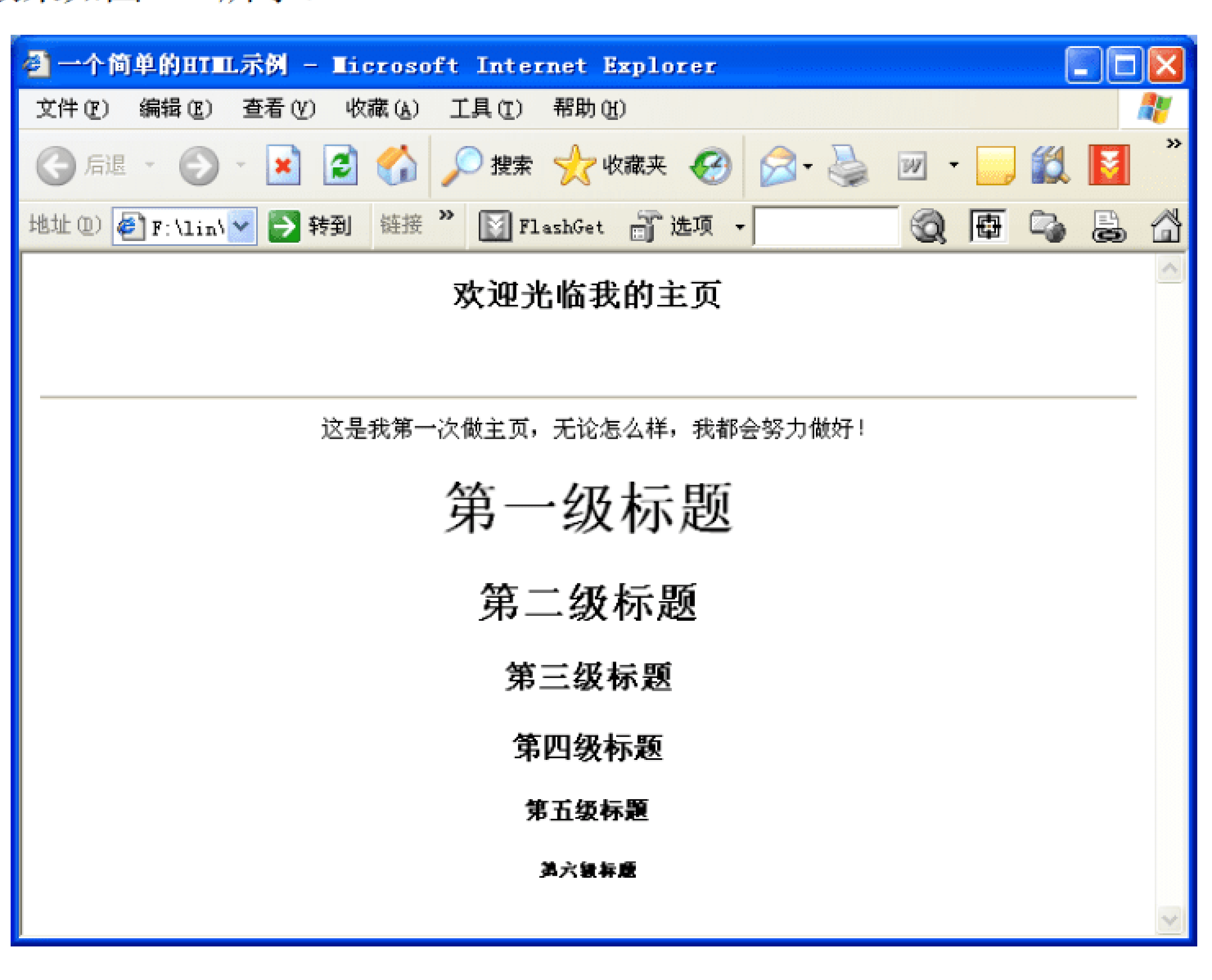


图 1-2　标题效果

在 HTML 语言规范里，每当浏览器窗口被缩小时，浏览器会自动将右边的文字转折至下一行。所以，网页制作者对于自己需要断行的地方，应加上
标签。

文件段落的开始由<P>来标记，段落的结束由</P>来标记，</P>是可以省略的，因为下一个<P>的开始就意味着上一个<P>的结束。<P>标签还有一个属性 ALIGN，它用来指明字符显示时的对齐方式，一般可选值有 CENTER、LEFT、RIGHT 这 3 种。

水平线段<HR>标签可以在屏幕上显示一条水平线，用以分割页面中的不同部分。<HR>有 4 个属性：size 控制水平线的宽度；width 控制水平线的长，用占屏幕宽度的百分比或像素值来表示；align 控制水平线的对齐方式，有 LEFT、RIGHT 和 CENTER 这 3 种；noshade 控制线段无阴影属性，并为实心线段。例如<HR size=3　width=88%　align=center　noshade>，其显示效果如图 1-3 所示。



图 1-3　水平线段效果

HTML 提供设置字号大小的是 FONT 标签，FONT 有一个属性 SIZE，通过指定 SIZE 属性就能设置字号大小，而 SIZE 属性的有效值范围为 1—7，其中默认值为 3，另外，也可以在 SIZE 属性值之前加上＋、－字符，来指定相对于字号初始值的增量或减量。此外，FONT 还提供了定义字体的功能，用 FACE 属性来完成这个工作。FACE 的属性值可以是本机上的任一字体类型，但要注意：只有对方的计算机中也装有相同的字体才可以在他的浏览器中出现预先设计的风格。设置方式如<font face=“字体”>。

为了让文字富有变化，或者为了刻意强调某一部分，HTML 提供了一些标签产生这些效果，现将常用的标签列举如下：

<B>	</B>	粗体
<I>	</I>	斜体
<U>	</U>	加下划线
<TT>	<TT>	打字机字体
<BIG>	</BIG>	大型字体
<SMALL>	</SMALL>	小型字体
<EM>	</EM>	表示强调，一般为斜体
<STRONG>	</STRONG>	表示特别强调，一般为粗体
<CITE>	</CITE>	用于引证、举例，一般为斜体

文字颜色设置格式如下：

```
<font color=color_value>…</font>
```



这里的颜色值可以是一个十六进制数(用#作为前缀)，也可以是颜色名称，如：

```
 Black = "#000000"        Green = "#00FF00"
Red    = "#FF0000"        Blue = "#0000FF"
 <font face=宋体  color=red>    <H1>  字体   </H1>   </font>
 <font face=华文新魏  color=#00ff00>    <H1>  字体    </H1>   </font>
 <font face=楷体_GB2312    color=#0000ff>    <H1>  字体    </H1>   </font>
```

读者可以亲自编辑相应 HTML 页面并浏览其效果。

2. 列表

列表分无序号列表和序号列表，无序号列表使用的是一对标签
，每一个列表项前再使用<LI>。其结构如下所示：

```
<UL>
 <LI>第一项
    <LI>第二项
    <LI>第三项
</UL>
```

效果如图 1-4 所示。

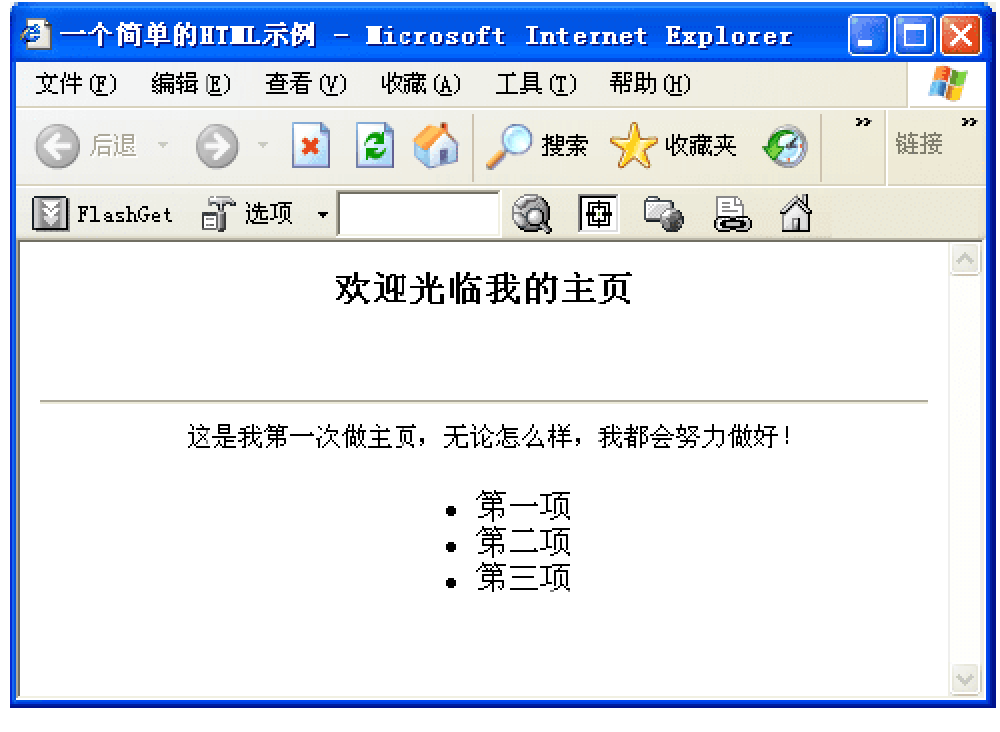


图 1-4　无序号列表效果

序号列表和无序号列表的使用方法基本相同，它使用标签<OL></OL>，每一个列表项前使用<LI>。每个项目都有前后顺序之分，并用数字标示。其结构如下所示：

```
        <OL>
        <LI>第一项
   <LI>第二项
   <LI>第三项
  </OL>
```

效果如图 1-5 所示。



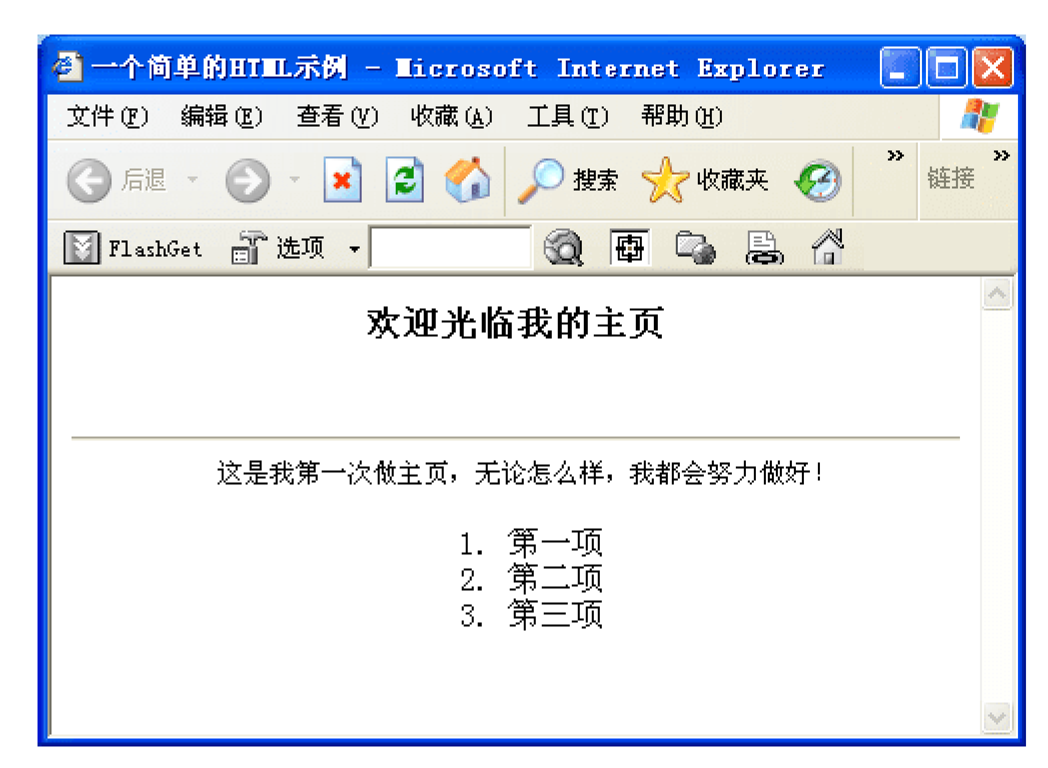


图 1-5　序号列表效果

3.　多媒体效果

在 HTML 网页中，可以通过标签插入图像、播放音乐和播放视频等来展示网页的多媒体效果。超文本支持的图像格式有 GIF、JPEG 等。插入图像的标签是<IMG>，属于单标签，其格式为：

```
<IMG SRC="图像文件地址">
```

其中图像文件的地址即可以是 Web 服务器的本机地址，也可以是一网络地址，但要确保该地址存在相应的图片文件。

需要在网页中嵌入音乐时可以将音乐做成一个链接，如：

```
<A  HREF="音乐地址">乐曲名</A>
```

当鼠标移动至乐曲名上方时，鼠标会由箭头变成小手，这时单击可以下载或直接播放相应的音乐，这种方式常见于音乐下载网站之中。如果希望一进入某个网页页面时，不用任何操作就能听见音乐立刻响起，则可以采用自动载入音乐方式，其基本语法为：

```
<EMBED SRC="音乐文件地址">
```

下面请看相应示例 HTML 文本：

```
<HTML>
 <HEAD>
  <TITLE>一个简单的 HTML 示例</TITLE>
 </HEAD>
 <BODY>
  <CENTER>
   <H3>欢迎光临我的主页</H3>
   <BR>
   <HR>
```



```
    <FONT SIZE=2>  这是我第一次做主页，无论怎么样，我都会努力做好！
    </FONT>
    <p>
     <IMG SRC="d.gif">
    </p>
    <p>
     <A HREF="wyc.mp3"> 忘忧草-周华健</A>
     <EMBED SRC="wyc.mp3" hidden>
    </p>
   </CENTER>
  </BODY>
</HTML>
```

其浏览效果如图 1-6 所示。



图 1-6　多媒体效果

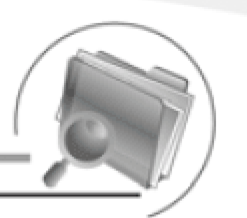
上述页面打开时，即可听见音乐响起，因为 HTML 文本中有这么一行代码<EMBED SRC="wyc.mp3" hidden>，另外还可看见两个人在月亮之上翩翩起舞，其实它只是一 gif 动画图片而已。需要注意的是，上述动画图片和音乐文件的地址都是采用本机的相对地址形式，表明与 HTML 文件同处一相同目录之下。

同理，借助<A></A>和<EMBED>标签，还可以在网页中实现视频的播放，比如将视频文件做成一个链接的方法如下：

```
<A   HREF=" 视频地址">视频名称</A>
```

而自动载入视频亦与音乐的播放一样，可以使用 EMBED 标签播放，格式如下：

```
<EMBED SRC= “视频文件地址” >
```



4. TABLE 表格

下面简单介绍表格的基本结构、表格的标题、表格的尺寸设置、表格内文字的对齐、布局、跨多行、多列的表元、表格的颜色等内容。

表格的基本结构可以通过下述标签来定义：

```
<table>...</table>      定义表格
 <caption>...</caption>   定义标题
  <tr>     定义表行
  <th>     定义表头
<td>     定义表元(表格的具体数据)
```

表格标题的位置，可由 ALIGN 属性来设置，其位置可以是表格上方或表格下方。下面为表格标题位置的设置格式。

设置标题位于表格上方：

```
<caption align=top> ... </caption>
```

设置标题位于表格下方：

```
<caption align=bottom> ... </caption>
```

一般情况下，表格的总长度和总宽度是根据各行和各列的总和自动调整的，如果要直接固定表格的大小，可以使用下列方式：

```
<table width=n1 height=n2>
```

width 和 height 属性分别指定表格一个固定的宽度和长度，n1 和 n2 可以用像素单位来表示，也可以用百分比(与整个屏幕相比的大小比例)来表示。

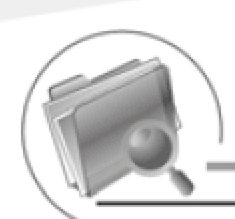
边框是用 border 属性来体现的，它表示表格的边框厚度和框线。将 border 设成不同的值，就会有不同的边框效果。

如：

```
<table border=10   width=250>
 <caption>定货单</caption>
<tr><th>苹果</th><th>香蕉</th><th>葡萄</th>
<tr><td>200 公斤</td><td>200 公斤</td><td>100 公斤</td>
</table>
```

格与格之间的线为格间线，它的宽度可以使用<TABLE>中的 CELLSPACING 属性加以调节。格式是：

```
<TABLE   CELLSPACING=#>          #表示要取用的像素值
```



例：

<table border=11　cellspacing=15>

还可以在<TABLE>中设置 CELLPADDING 属性，用来规定内容与格线之间的宽度。格式为：

<TABLE　CELLPADDING=#>　　　#表示要取用的像素值

例：

<table border=3　cellpadding=5>

表格中数据的排列方式有两种，分别是左右排列和上下排列。左右排列可以用 ALIGN 属性来设置，而上下排列则由 VALIGN 属性来设置。

其中左右排列的位置可分为 3 种：居左(left)、居右(right)和居中(center)；而上下排列基本上比较常用的有 4 种：上齐(top)、居中(middle)、下齐(bottom)和基线(baseline)。其设置示意如下所示：

<tr align=#>　<th align=#>　<td align=#>　　#=left, center, right
<tr valign=#>　<th valign=#>　<td valign=#>　　#=top，middle，bottom，baseline

在表格中，既可以对整个表格填入底色，也可以对任何一行、一个表元使用背景色。其设置为：

表格的背景色彩　<table bgcolor=#>
行的背景色彩　　<tr bgcolor=#>
表元的背景色彩　<th bgcolor=#>　或　<td bgcolor=#>

其中，#=rrggbb，为一十六进制 的 RGB 色彩编码，或者也可以是下列预定义的色彩名称：Black，Olive，Teal，Red，Blue，Maroon，Navy，Gray，Lime，Fuchsia，White，Green，Purple，Silver，Yellow，Aqua 等。

1.1.4　Web 结构

早期互联网刚刚兴起时，很多公司网站或个人主页都是用基本的 HTML 语言编写的，然后再通过称为 Web 服务器的软件来进行发布。其结构如图 1-7 所示。



图 1-7　Web 结构



从图 1-7 可见，Web 结构主要由两个部分组成：提供 Web 页面信息服务的 Web 服务器端和向 Web 服务器端发出信息内容浏览请求的客户端浏览器。服务器端网站存放包含各种形态的多媒体信息网页，它们通过 Web 服务器对广大网络用户进行发布(即用户可以通过 HTTP 协议来获取)，因此 Web 服务器通常也称 HTTP 服务器，常见的 Web 服务器有 Apache HTTP 服务器，Netscape 的企业服务器(NES)，iPlanet Web 服务器和微软的 IIS(Internet 信息服务器)等。客户端主要包含了各种可以浏览网页内容的浏览器软件，目前比较常用的浏览器软件有 Windows 操作系统附带的 Internet Explorer(简称 IE)浏览器和火狐 FireFox 等。在上网浏览网站时，浏览器向对方的 Web 服务器发送一个请求，Web 服务器接收到请求以后会搜索相应的页面，若搜索到相应页面，Web 服务器就会通过 HTTP 协议向客户端浏览器发送该页面，客户端浏览器获取该页面后就进行解析显示，若该页面不存在，则客户端会收到相应的报错信息。

下面对与 Web 结构相关的几个重要概念作一详细解释。

1. 客户端与浏览器

要打开和浏览网络上的网页文件，必须通过浏览器软件，如 IE、FireFox 等，而使用浏览器打开网页的这一端计算机系统，称为客户端，因为服务器网站为上网用户提供网页信息服务，因此用户也可被称为客户。

浏览器的功能主要是获取网页文件(主要是 HTML)并解析和显示文件中的内容，若 HTML 中同时含有客户端执行的脚本语言，例如 VBScript 或 JavaScript，则浏览器同样会对其进行解释的操作，最后将网页的执行结果显示在用户的浏览器窗口中。

2. 服务器端与服务器

与客户端相比，提供浏览网页服务的一方称作“服务器端”，而用来放置这些网页信息的计算机，则称为服务器。

服务器的功能并不只是单纯地存放网页信息，任何可提供网络服务的计算机都可以是服务器，例如提供网页信息的称为 Web 服务器，而提供文件上传与下载功能的则称为 FTP 服务器。

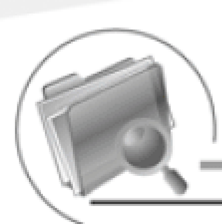
3. 通信协议

在网络上要能彼此互通信息就必须遵循一定的交流规则，这些交流规则即是所谓的通信协议(Protocol)，如表 1-1 所示为较常用的应用层通信协议。

表 1-1　应用层通信协议

通 信 协 议	说　明
http	超文本传输协议，主要用来传送文字、图片、声音等多媒体类型的数据
ftp	文件传输协议，用来上传文件至远程主机，或从远程主机下载文件至本地计算机
smtp 和 pop3	常见的邮件通信协议，主要用来建立邮件发送和接收服务
telnet	远程登录协议，可以用来登录 BBS 系统

只有当通信双方都使用相同的协议，它们之间才可以顺利交流。



4. 全球资源定位器——URL

URL 的英文全称为 Uniform Resource Locator(全球资源定位器)，当用户想要打开位于远程网站主机上的网页时，必须指定其 URL 位置，也就是通常所讲的网址，如下所示为一个 URL 地址。

http://www.sina.com.cn/index.html

URL 的功能在于告诉浏览器资源相关信息、资源所在网络地址及所使用的通信协议，上例采用 http 协议访问 www.sina.com.cn 服务器上的 index.html 页面信息，页面信息不一定是 html 的，也可以是如图 1-8 所示的 jsp 类型页面。




图 1-8　中央电视台的官方网站 URL

总之，当浏览器接受了用户输入的一个网址后，便会根据其中所提供的信息，向服务器提出网页浏览请求。此外，URL 中的通信协议可以是其他类型的，如：

ftp://www.tsinghua.edu.cn/materials.rar

该 URL 表示采用 ftp 协议下载 www.tsinghua.edu.cn 服务器上的 materials.rar 压缩包资源。

1.2　JSP 概述

在介绍 JSP 技术之前，有必要先对 Java 语言和 Servlet 技术进行简要说明。

1.2.1　Java 语言

Java 是由美国 Sun 公司开发的支持面向对象程序设计的语言，它最大的优势是借助于虚拟机机制实现的跨平台特性，实现所谓的“Write once，run everywhere”，使得移植工作变得十分容易！也正因为此，使得 Java 迅速流行起来，成为一种深受广大开发者喜欢的编程语言，目前，随着 J2ME、J2SE 和 J2EE 的发展，Java 已经不仅仅是一门简单的计算机开发语言了，它已经拓展发展出一系列的业界先进技术。

目前 Java 已被业界广泛接受，Microsoft、IBM、DEC、Adobe、SiliconGraphics、HP、Oracle、Toshiba、Netscape 和 Apple 等大公司均早已购买了 Java 的许可证。Microsoft 还在其 Web 浏览器中增加了对 Java 的支持。另外，众多的软件开发商也开发了许多支持 Java 的软件产品，如美国 Borland 公司的 JBuilder，蓝色巨人 IBM 的 Eclipse 和 Visual Age for Java，太阳公司 Sun





的NetBeans 与Sun Java Studio 5以及BEA公司的WebLogic Workshop等。数据库厂商如Oracle，Sybase也都在开发支持HTML和Java的CGI(Common Gateway Interface)，甚至Oracle公司还将自己的数据库产品用Java来进行开发。在以网络为中心的计算时代，不支持HTML和Java，就意味着应用程序的应用范围只能局限于同质的环境。Intranet正在成为企业信息系统最佳的解决方案，它的优点表现在：便宜、易于使用和管理。用户不管使用何种类型的机器和操作系统，界面都是统一的Web浏览器，而数据库、Web页面(HTML和用Java编的JSP、Servlet等)、中间件(Java Bean或Enterprise Java Bean等)则存在WWW和应用服务器上。开发人员只需维护一个软件版本，管理人员省去了为用户安装、升级客户端以及培训人员之繁琐，用户则只需一个操作系统，一个Internet浏览器(当然，浏览器并不限定就要用微软的Internet Explorer，读者也可以考虑采用FireFox，Netscape，Opera等等)就可以运行了。这就是现在常说的B/S(浏览器/服务器)模式，它与C/S(客户/服务器)模式的显著不同就在于其是“瘦客户端”的，这样就使得程序运行对客户端的要求降至很低的水平，一般将C/S模式开发的软件称为两层架构的，而B/S模式的软件为三层(或多层)架构的，J2EE系列技术就是致力于帮助客户构建多层架构的应用，而JSP是J2EE中非常重要的一项技术。

1.2.2　Servlet 技术

Servlet是位于Web服务器端的Java应用程序，与传统的从命令行启动的Java应用程序不同，Servlet由Web服务器进行加载，该Web服务器必须包含支持Servlet的Java虚拟机。Java Servlet是Java中非常重要的技术之一，更是Web应用程序的基础。1997年3月JavaSoft正式推出了Servlet 1.0标准版，至今已经发展到2.5版。Servlet作为纯Java对象驻留在服务器端，通过监听客户端的请求来执行相应的运算逻辑。事实上Servlet不仅仅用于Web/HTTP环境，它还支持RMI和CORBA等体系结构，不过应用最广泛的还是Web服务器页面的处理。

Servlet具备如下优点：

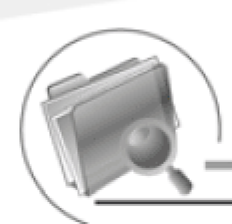
- 可移植性

Servlet都是用纯Java语言来开发的，所以编写的Servlet程序在Windows、UNIX和Linux等操作系统上都可以运行，只要在该操作系统上安装了带有符合规范的Servlet引擎的服务器即可。借助Servlet的优势，程序员可以真正地实现“一次编写，随处运行”而不用作代码的修改。同样，由于Servlet是在服务器端运行的，所以程序员也不需要考虑客户端使用何种版本的Java运行环境(JRE)。

- 强大的功能

由于Servlet是一个Java应用，所以Servlet能够发挥Java API(应用程序编程接口)的强大威力。通过这些API，可以实现诸如网络和URL存取、多线程服务、使用JDBC进行数据库访问、使用JNDI访问目录服务、远程方法调用(RMI)、对象序列化和分布式服务器组件EJB等的应用。

API的应用，使Servlet具有比过去的公共网关接口(CGI)以及ASP和PHP等技术所无法比



拟的强大功能。

⦿ 性能

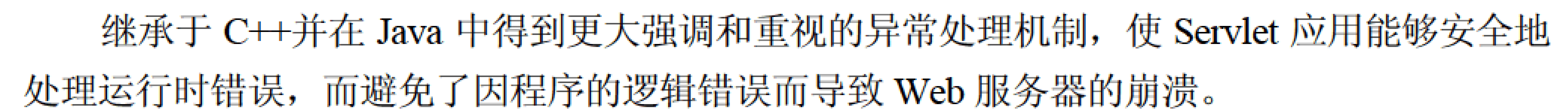
Servlet 一旦被加载运行，它的实例将驻留在内存中。当接收到客户端的请求时，服务器会调用 Servlet 来处理请求。如果同时有多个相同的客户端请求，Servlet 会启动不同线程来分别处理。这种使用线程而非进程的方法使 Servlet 的性能大大优于过去的 CGI 方式。

⦿ 安全性

由于 Servlet 是一个 Java 应用，所以它同样是强类型的，可以避免很多由于类型转换而产生的内存错误问题。

Java 的垃圾收集功能使得 Servlet 不用考虑如何释放相应的内存资源问题，从而简化了内存管理，同时也减少了内存错误发生率。

Java 摒弃了 C/C++中的指针，避免了 Servlet 对内存的直接访问，使 Servlet 应用的安全性、稳定性和坚固性得到很大的提升。



继承于 C++并在 Java 中得到更大强调和重视的异常处理机制，使 Servlet 应用能够安全地处理运行时错误，而避免了因程序的逻辑错误而导致 Web 服务器的崩溃。

1.2.3 JSP 技术

Java Server Page(简称 JSP)是基于 Java 的技术，用于创建可支持跨平台以及跨 Web 服务器的 Web 服务器端应用程序(即所谓的动态网页)。它是由 Sun Microsystem 公司倡导，由多家公司合作而建立的一种动态网页技术标准。它与目前同样流行的 ASP 技术、ASP.NET 技术是相同性质的、同一层次的，它们在网站的建设中所起的作用是一样的，但是 JSP 技术与这两种技术相比，有着十分突出的优越性，因为 JSP 技术有 J2EE 平台支持，发展前途不可限量。相对于传统的网页制作技术而言，JSP 技术有着明显的优点：JSP 不像 CGI、ISAPI 和 NSAPI 一样难于编写和维护；同时不像 PHP 一样只能适用于中小流量的网站，而且具有良好的扩充性；也不像 ASP 一样受到跨平台的限制(只能运行于 Microsoft 公司开发的 IIS 和 PWS 上)。JSP 体现了当今最先进的网站开发思想，它有诸多的特点。

1. JSP 技术的特点

⦿ 将内容的生成和显示分离

用 JSP 技术，Web 页面开发人员可以使用 HTML 或者 XML 标识来设计和格式化最终页面，并使用 JSP 标识或者小脚本来生成页面上的动态内容(内容是根据请求变化的，例如请求账户信息或者特定的一瓶酒的价格等)。生成内容的逻辑被封装在标识和 JavaBeans 组件中，并且捆绑在脚本中，所有的脚本在服务器端运行。由于核心逻辑被封装在标识和 JavaBeans 中，所以 Web 管理人员和页面设计者，能够编辑和使用 JSP 页面，而不影响内容的生成。在服务器端，JSP 引擎解释 JSP 标识和脚本，生成所请求的内容(例如，通过访问 JavaBeans 组件，使用 JDBC 技术访问数据库或者包含文件)，并且将结果以 HTML(或者 XML)页面的形式发送回浏览器。





这既有助于作者保护自己的代码，又能保证任何基于 HTML 的 Web 浏览器的完全可用性。

⊙ 利用可重用组件

绝大多数 JSP 页面依赖于可重用的、跨平台的组件(JavaBeans 或者 Enterprise JavaBeans 组件)来执行应用程序所要求的复杂的处理。开发人员能够共享和交换执行普通操作的组件，或者使得这些组件为更多的使用者和客户团体所使用。基于组件的方法加速了总体开发过程，并且使得各种组织在他们现有的技能和优化结果的开发努力中得到平衡。

⊙ 采用标识简化开发

由于不是所有的 Web 页面开发人员都熟悉脚本语言，所以 Java Server Pages 技术封装了许多功能，这些功能是在与 JSP 相关的 XML 标识中生成动态内容所需要的。标准的 JSP 标识能够访问和实例化 JavaBeans 组件、设置或者检索组件属性、下载 Applet，以及执行用其他方法难于编码且耗时的功能。通过开发和定制标识库，可以扩展 JSP 技术。所以，第三方开发人员和其他人员可以为常用功能创建自己的标识库，这也使得 Web 页面开发得以一定简化。

⊙ 强大的可伸缩性

从只有一个小的 jar 文件就可以运行 servlet/jsp 到进行 Application 事务处理、消息处理；从一台服务器到无数台服务器进行集群和负载均横，Java 显示了它强大的可伸缩性。

⊙ 多样化和功能强大的开发工具支持

这一点与 ASP 很像，Java 已经有了许多非常优秀的开发工具而且有许多可以免费得到，并且其中的许多已经可以顺利的运行于多种平台之下。如果细心的使用它们会发现比自己第一面看到它们时的功能要强大的多。

⊙ 平台无关

几乎所有平台，如 Linux、Mac、Windows、Unix、Solaris 等都支持 Java，JSP＋JavaBeans 几乎可以在所有平台下通行无阻。从一个平台移植到另外一个平台，JSP 和 JavaBeans 甚至不用重新编译，因为 Java 字节码都是标准的与平台无关的。作为 Java 平台的一部分，JSP 拥有 Java 编程语言“一次编写，各处运行”的特点。

⊙ 数据库连接

Java 中连接数据库的技术是 JDBC，Java 程序通过 JDBC 驱动程序与数据库相连，执行查询、提取数据等操作。Sun 公司还开发了 JDBC－ODBC Bridge，利用此技术 Java 程序可以访问带有 ODBC 驱动程序的数据库，目前大多数数据库系统都带有 ODBC 驱动程序，所以 Java 程序能访问诸如 Oracle、Sybase、MS SQL Server 和 MS Access 等数据库。此外，通过开发标识库，JSP 技术可以进一步扩展。第三方开发人员和其他人员可以为常用功能创建自己的标识库。这使得 Web 页面开发人员能够使用熟悉的工具和如同标识一样的执行特定功能的构件来进行工作。

JSP 技术可以很容易整合到多种应用体系结构中，以利用现存的工具和技巧，并且能扩展到支持企业级的分布式应用中。作为采用 Java 技术家族的一部分，以及 Java 2(企业版体系结构)的一个组成部分，JSP 技术能够支持高度复杂的基于 Web 的应用。由于 JSP 页面的内置脚本语言是基于 Java 的，而且所有的 JSP 页面都被编译成为 Java Servlets，所以 JSP 页面具有 Java 技

术的所有好处，包括健壮的存储管理和安全性。作为 Java 平台的一部分，JSP 拥有 Java 编程语言“一次编写，各处运行”的特点。

利用 JSP 技术，动态信息由 JSP 页面来表现，JSP 页面由安装在 Web 服务器或者使用 JSP 的应用服务器上的 JSP 引擎执行。JSP 引擎接受客户端对 JSP 页面的请求，并且生成 JSP 页面作为对客户端的响应。JSP 页面通常被编译成为 Java Servlets，这是一个标准的 Java 扩展。页面开发人员能够访问全部的 Java 应用环境，以利用 Java 技术的扩展性和可移植性。当 JSP 页面第一次被调用时，如果它还不存在，就会被编译成为一个 Java Servlets 类，并且存储在服务器的内存中。这就使得在接下来的对该页面的调用中，服务器会有非常快的响应(这避免了 CGI－BIN 为每个 HTTP 请求生成一个新的进程的问题)。

JSP 页面可以包含在多种不同的应用体系结构或者模型中，可以用于由不同协议、组件和格式所组成的联合体中。基于 JSP 的动态信息发布技术是一个开放的、可扩展的建立动态 Web 页面的标准。不论采用什么创建工具，开发人员都可以使用 JSP 页面来创建可移植的 Web 应用，在不同的 Web 应用服务器上运行。

2. JSP 的运行原理

首先客户端发出请求，Web 服务器接收到请求之后，Web 服务器对 JSP 代码进行操作必须经历 3 个过程：第一过程是代码转化，具体内容是用 JSP 引擎把 JSP 代码、相关组件、Java 脚本以及 HTML 代码转化成为 Servlet 代码；第二过程是编译，具体内容是用 Java 编译器对 Servlet 代码进行编译；第三过程是执行编译文件，编译文件的执行是由 Java 虚拟机完成的。在上述 3 个过程依次完成后，再由 Java 虚拟机将执行结果返回给 Web 服务器，并最终返回给客户端，这便是一个完整的 JSP 执行过程。 由上述情况可知，JSP 的执行必须同时具备 3 个条件：JSP 引擎、Web 服务器以及 Java 虚拟机。

(1) JSP 引擎：JSP 引擎在 JSP 运行中起到将 JSP 代码转化成 Servlet 代码的作用，并能够判断是否需要对 Servlet 代码进行编译以及重新编译，最后向 Java 虚拟机发出通知。

(2) Web 服务器：此类 Web 服务器必须支持 JSP 的运行，在接收到客户端的请求后，经过各种处理，将 JSP 执行的结果返回到客户端。

(3) Java 虚拟机：所谓 Java 虚拟机是指能够运行 Java 代码的假想计算机，也即是一种能把 Java 程序代码翻译成机器码的程序。在 JSP 执行过程中，能够作为 Java 编译器完成对 Servlet 代码的编译，并能执行编译后的字节码。

JSP 的具体执行过程如下：

(1) 通过客户端的浏览器，以超文本(HTML)形式通过表单(FORM)向 Web 服务器提出请求。

(2) 服务器得到客户端的请求后，由 Web 服务器上的 JSP 引擎把 JSP 代码、相关组件、Java 脚本以及 HTML 代码，转化成为 Servlet 代码。

(3) 接着由 JSP 引擎调用 Web 服务器端的 Java 编译器对 Servlet 代码进行编译。



(4) 最后，由 Java 虚拟机执行编译后的字节码文件，并把执行结果以标准 HTML 页面的形式返回给客户端。

图 1-9 简要地刻画了 JSP 运行原理。

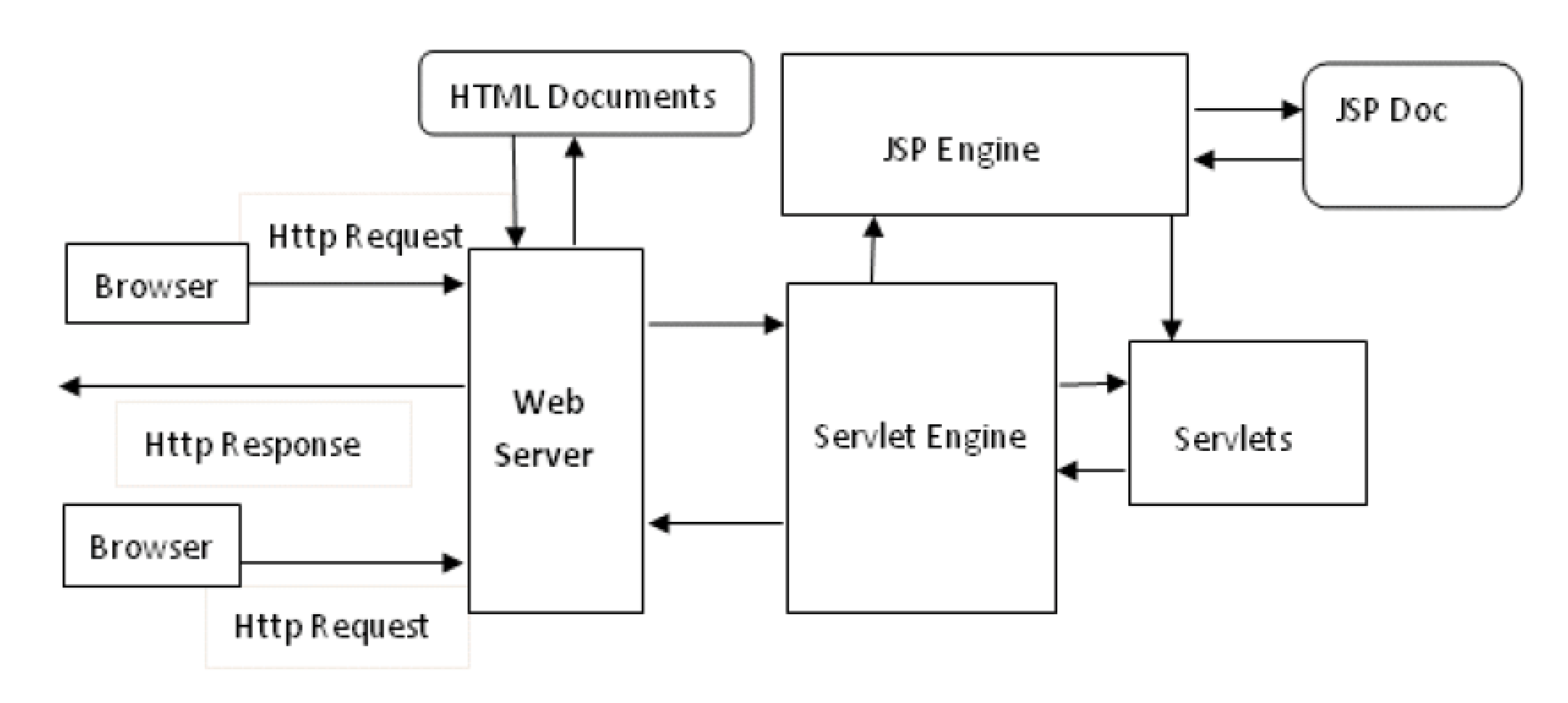


图 1-9　JSP 运行原理



1.3　习题

1.3.1　填空题

1. 常用的 Web 服务器有：__________、__________和__________等。
2. JSP 使用__________语言实现动态显示。

1.3.2　选择题

1. 以下文件名后缀中，只有(　　　)不是静态网页的后缀。

A. .html　　B. .htm　　C. .jsp　　D. .shtml

2. 以下文件名后缀中，只有(　　　)不是动态网页的后缀。

A. .jsp　　B. .xml　　C. .aspx　　D. .php

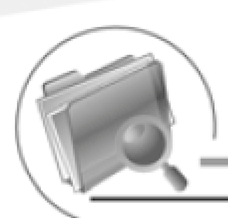
3. 下列描述中，只有(　　　)是错误的。

A. JSP 提供了多种语言支持。

B. JSP 提供了多种平台支持。

C. JSP 采取编译执行的方式，极大地提高了运行性能。

D. JSP 提供跨平台支持，也可以在 UNIX 下执行。



4. Java 程序可以在以下哪些平台上运行(　　)(多选)。

A.　Windows　　B. Linux　　C. UNIX　　D. DOS

1.3.3　问答题

1. 简述 Web 结构及其各组成部分？
2. 请解释说明 JSP 的具体执行过程。



第 2 章 JSP 运行环境和开发环境

学习目标

在第 1 章中简单介绍了 HTML 和 JSP 的用途和功能。在开始学习使用 JSP 之前，需要先来了解一下运行 JSP 所需的环境。对于开发 JSP 来说，虽然一个简单的记事本就可以，不过为了调试方便，还是需要有一个集成多种功能的开发工具。使用 Dreamweaver 来制作页面，而使用 Eclipse 进行 Java 代码的开发调试是目前比较流行的一种开发工具组合。本章学习目标正是 JSP 的运行和开发环境。

本章重点

- JSP 的运行环境
- Tomcat 的安装和配置
- Eclipse 的安装
- JSP 的开发方式

2.1 运行环境

本节将分别介绍运行 JSP 页面所需的客户端和服务器端环境。简要介绍 JDK 和 Tomcat 的安装与配置。

2.1.1 JSP 客户端运行环境

作为一种 Internet Web 应用开发技术，JSP 对于客户端并没有特殊要求。理论上对于任何支持 HTML 规范的 JSP 页面，不管客户端使用的是 Windows、UNIX、Linux 还是 OS390 等操

作系统，只要安装了能够解析相应 HTML 规范的浏览器即可正常运行。常见的 Web 浏览器包括 Internet Explorer (IE)、Netscape Navigator、Mozilla、Opera、Maxthon 和 FireFox 等。

在实际开发过程中，由于各个浏览器对于部分 HTML 元素的解析方式不同，加上各个供应商对 HTML 规范又有各自不同的扩展，同一个 JSP 页面在不同的浏览器中可能会显示出细微的差别。另外不同的浏览器对于一些客户端动态技术，如 JavaScript 和 Java Applet 等支持程度不一样，也可能会产生不同效果，或者可能要安装一些支持软件如 JRE(Java Runtime Environment，Java 运行环境)等。因此，为了使 Web 应用能够满足最终用户的不同客户端需求，JSP 开发人员应该尽可能的只使用服务器端脚本，采用 HTML 规范。如果必须使用一些客户端脚本，也要尽可能地选用主流浏览器都能够支持的技术和语法，并在不同平台上加以测试。

本书的实例基本上只采用服务器端脚本，并在 Windows XP 中文专业版和 Internet Explorer (IE) 6.0 浏览器平台上测试通过。



2.1.2 JSP 服务器端运行环境

在服务器端，为了使 JSP 页面能够正常运行，需要 Web 服务器处理 Web 页面请求；需要 Java 运行环境来支撑对于 JSP 或 Servlet 的编译和运行；还需要 JSP 和 Servlet 容器来解析 JSP 页面和 Servlet 请求。由于 Sun 公司将 Java 技术公开作为一种开发的标准，所以业界提供了非常多的服务器选择，包括 IBM 的 WebSphere、BEA 的 WebLogic、Sun 的 iPlanet 以及开源社区 Apache 项目的 Tomcat 和 JRun 等。即使是微软的 Internet 信息服务(IIS)Web 服务器，也有一些第三方供应商提供了 JSP 和 Servlet 的支持组件，从而能够支持 JSP 的开发与部署。

WebSphere 和 WebLogic 是目前市场上占有率最大的两个 J2EE 服务器提供商，功能也非常强大，且都内置了 JDK 和 Web 服务器的支持。但由于强大的功能也导致其占用资源也非常庞大，另一方面其高昂的价格不仅让多数中小企业无法承受，而且与 Java 社区的开放性背道而驰。因此，目前日渐流行的服务器运行环境组合是 JDK+Apache HTTP Server+Tomcat。

JDK(Java Development Kit)是由 Sun 公司开发的 Java 开发工具包，目前其最新的稳定版本是 1.5.0，它提供了 Java 程序(包括 JSP、Servlet、Applet 和 JavaBean 等)的开发和运行环境。Tomcat 则是由 Apache 开源组织开发的一个符合 J2EE(Java 2 企业版，包括 JSP、Servlet 和 EJB 等技术)规范的一个 Web 应用服务器，最新的稳定版本是 5.5.9，而且在此版本中内置了 Apache HTTP Server 来处理 Web 请求。下面将简单介绍 JDK 1.5.0 和 Tomcat 5.5.9 的安装和配置。

2.1.3 JDK 安装

JDK 的安装程序可以从 Java 的官方网站 http://java.sun.com 获取，由于 Tomcat 5.5.9 要求 JDK 1.5.0 的支持，所以需要在网站上下载 JDK1.5.0 的安装程序。当前版本的 JDK 安装程序文件名为 jdk-1_5_0_04-windows-i586-p.exe。



下面以在 Windows XP 操作系统上安装 JDK 1.5.0 为例简要介绍一下如何安装 JDK。

(1) 打开 JDK 安装程序所在文件夹，双击 jdk-1_5_0_04-windows-i586-p.exe 文件执行安装程序。弹出如图 2-1 所示的 Installshield Wizard 对话框检查系统配置。

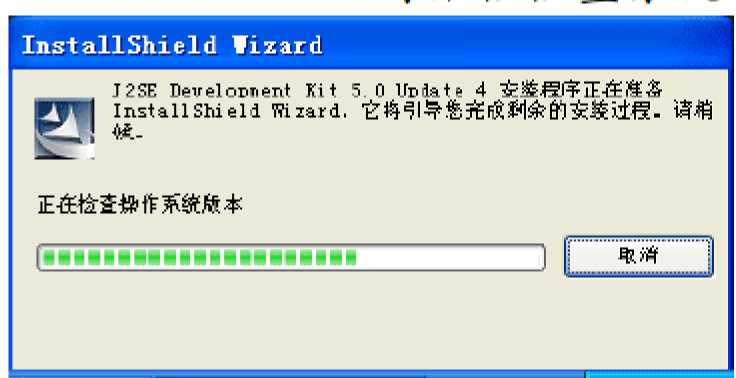
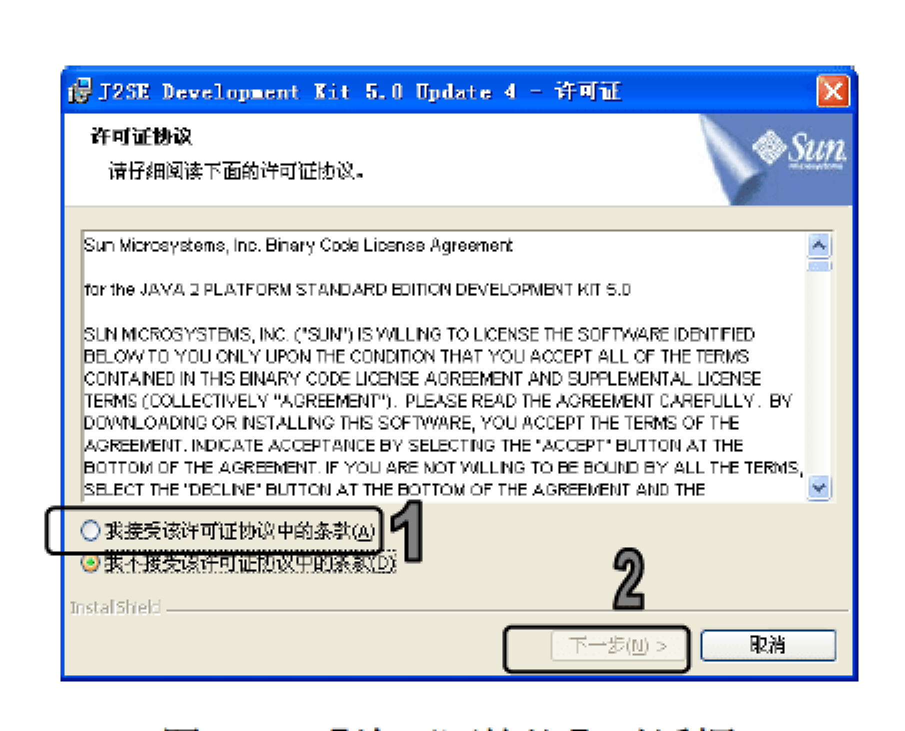


图 2-1　JDK 安装

(2) 完成系统配置检查后就可以进行后续安装了，首先弹出【许可证协议】对话框，如图 2-2 所示。

(3) 选择【我接受该许可证协议中的条款】单选按钮，然后单击【下一步】按钮后，打开提示用户选择要安装的可选功能的【自定义安装】对话框，如图 2-3 所示。



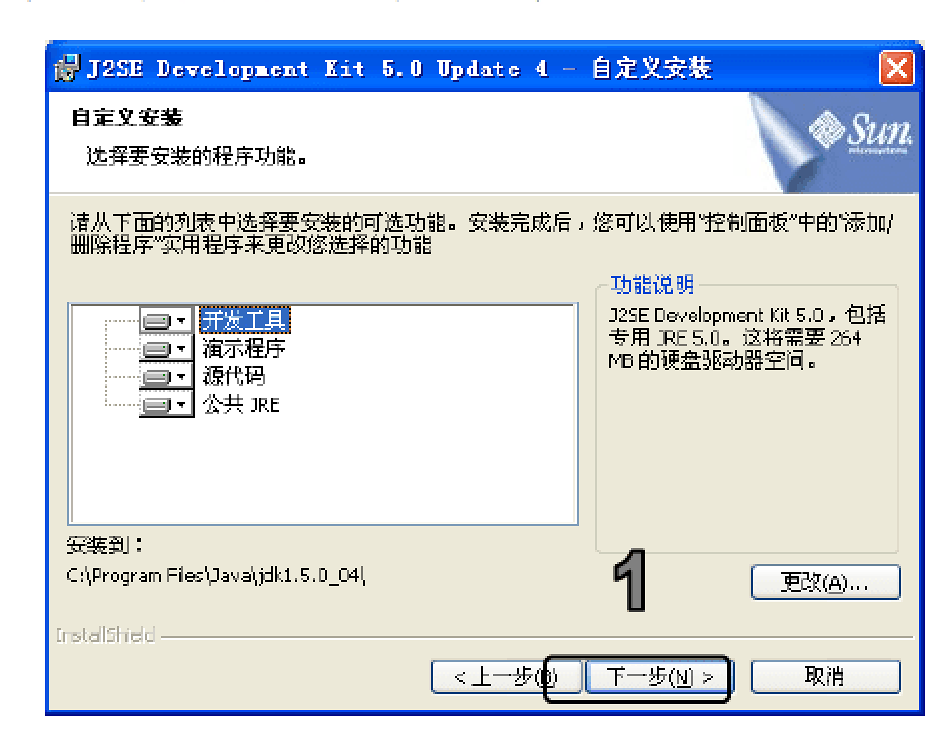


图 2-2　【许可证协议】对话框　　　　图 2-3　【自定义安装】对话框

(4) JDK 包括【开发工具】、【演示程序】、【源代码】和【公共 JRE】4 部分，除了【开发工具】是必须要安装的之外，其他几项均为可选的。单击这几项内容前面的下拉图标，即可显示安装选项，如图 2-4 所示。

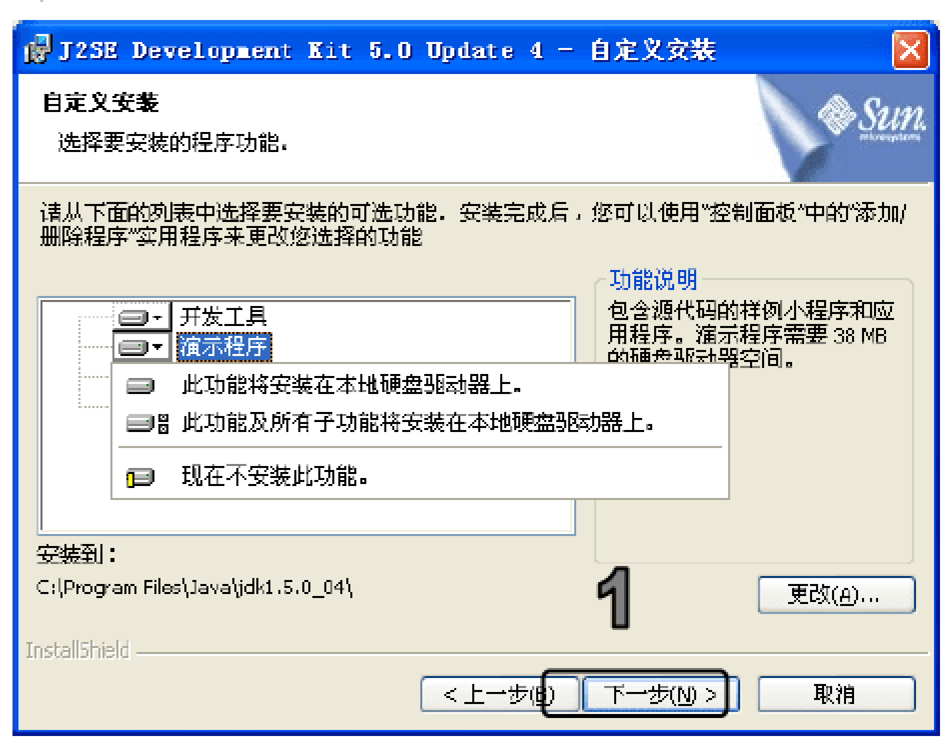
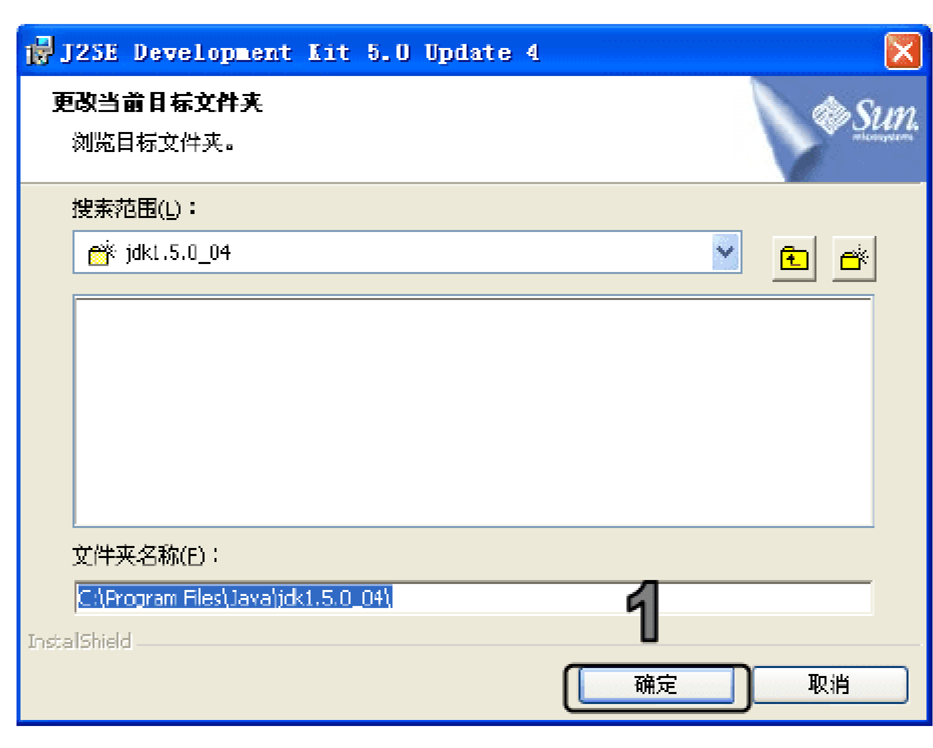


图 2-4　【自定义安装】对话框–选择可选功能



(5) 选定安装选项之后，可以通过单击【更改】按钮更改安装路径，如图 2-5 所示。修改目标文件夹后单击【确定】按钮返回上一对话框。

(6) 单击【自定义安装】对话框中的【下一步】按钮后，将弹出如图 2-6 所示的【进度】对话框显示当前的安装进度。



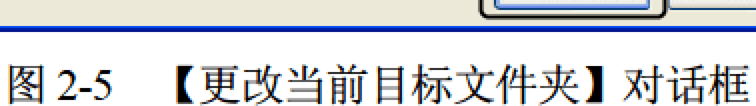


图 2-5 【更改当前目标文件夹】对话框

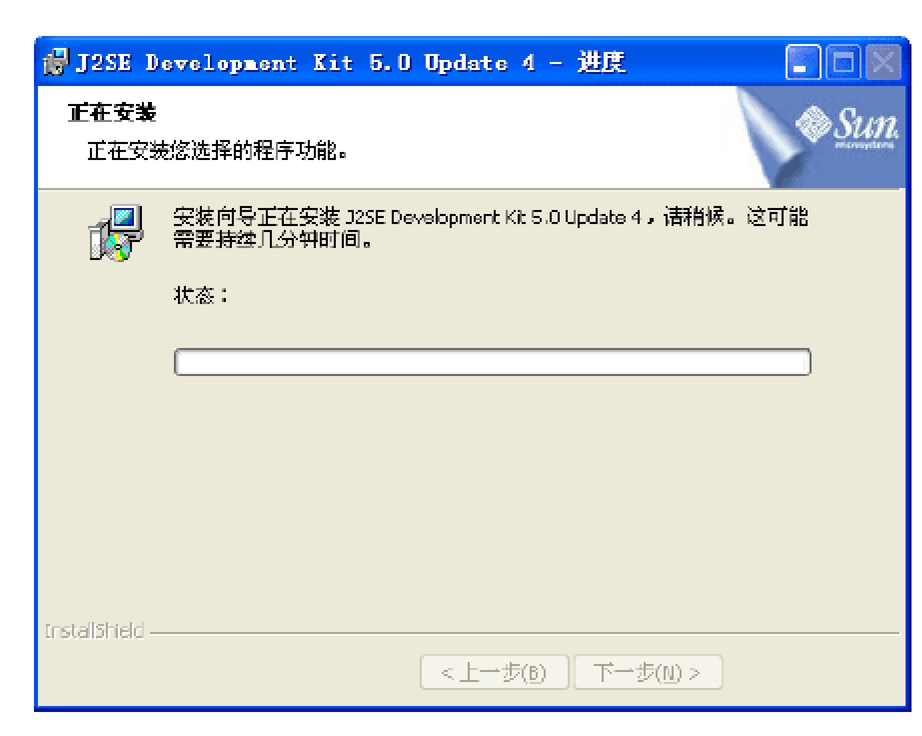


图 2-6 【进度】对话框



(7) 如果在【自定义安装】对话框中选择了安装【公共 JRE】(为方便 Tomcat 的安装，最好选择安装此项)，则会在【进度】对话框显示接近结束时自动弹出 JRE 的安装程序，如图 2-7 所示。这个 JRE 提供了对于多国语言的支持。同样可以通过单击【更改】按钮选择安装目标文件夹。

(8) 单击【下一步】按钮后，弹出注册 Java 插件的浏览器的【浏览器注册】对话框，如图 2-8 所示。如果系统中只有 Microsoft Internet Explorer 一种浏览器，则只出现一个选项，选中浏览器名前方的复选框之后，JRE 会在该浏览器中注册 Java 插件。这样，一些客户端的脚本或 Applet 就可以方便地运行了。

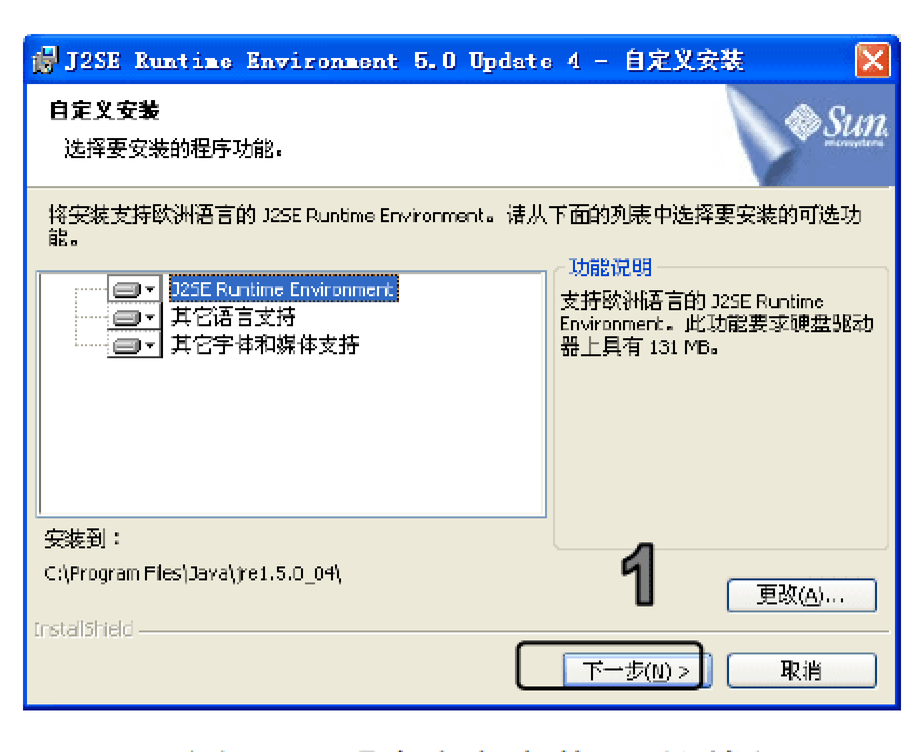


图 2-7 【自定义安装】对话框

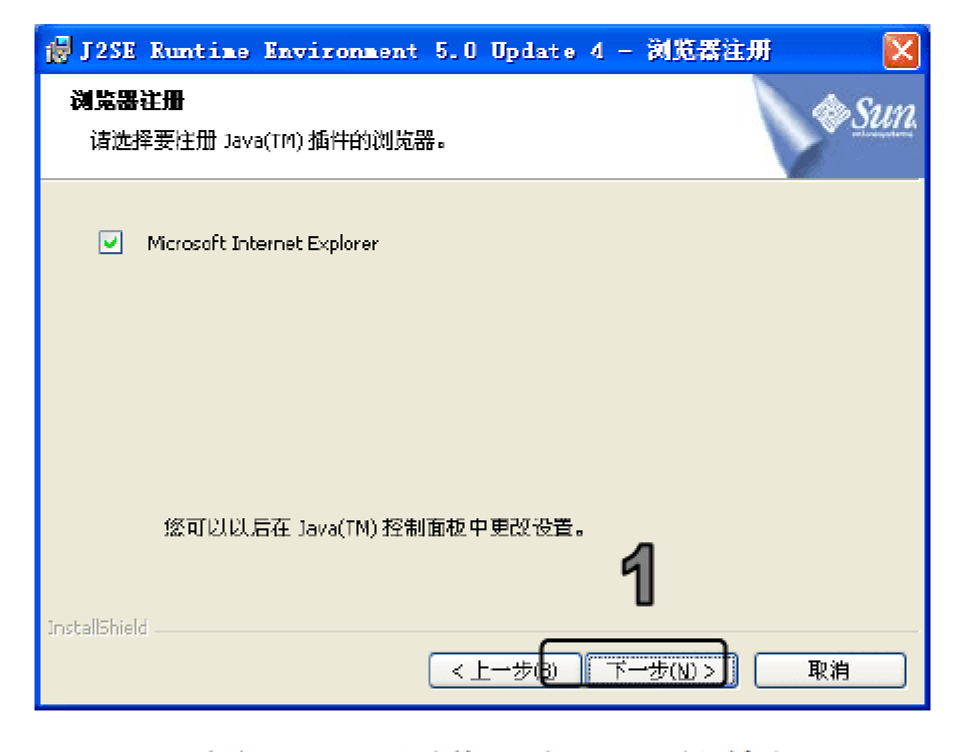


图 2-8 【浏览器注册】对话框

(9) 单击【下一步】按钮，弹出如图 2-9 所示【进度】对话框显示 JRE 的安装进度。

(10) JRE 安装完成后，自动关闭打开的对话框，同时提示整个 JDK 安装完成，如图 2-10 所示。单击【完成】按钮即可结束 JDK 的安装。



图 2-9　JRE 安装【进度】对话框



图 2-10　安装完成

(11) 单击系统的【开始】|【运行】命令，在【运行】对话框中输入 cmd，弹出命令行窗口。在窗口中输入“java”命令，如果显示如图 2-11 所示的信息，则说明已经成功安装。

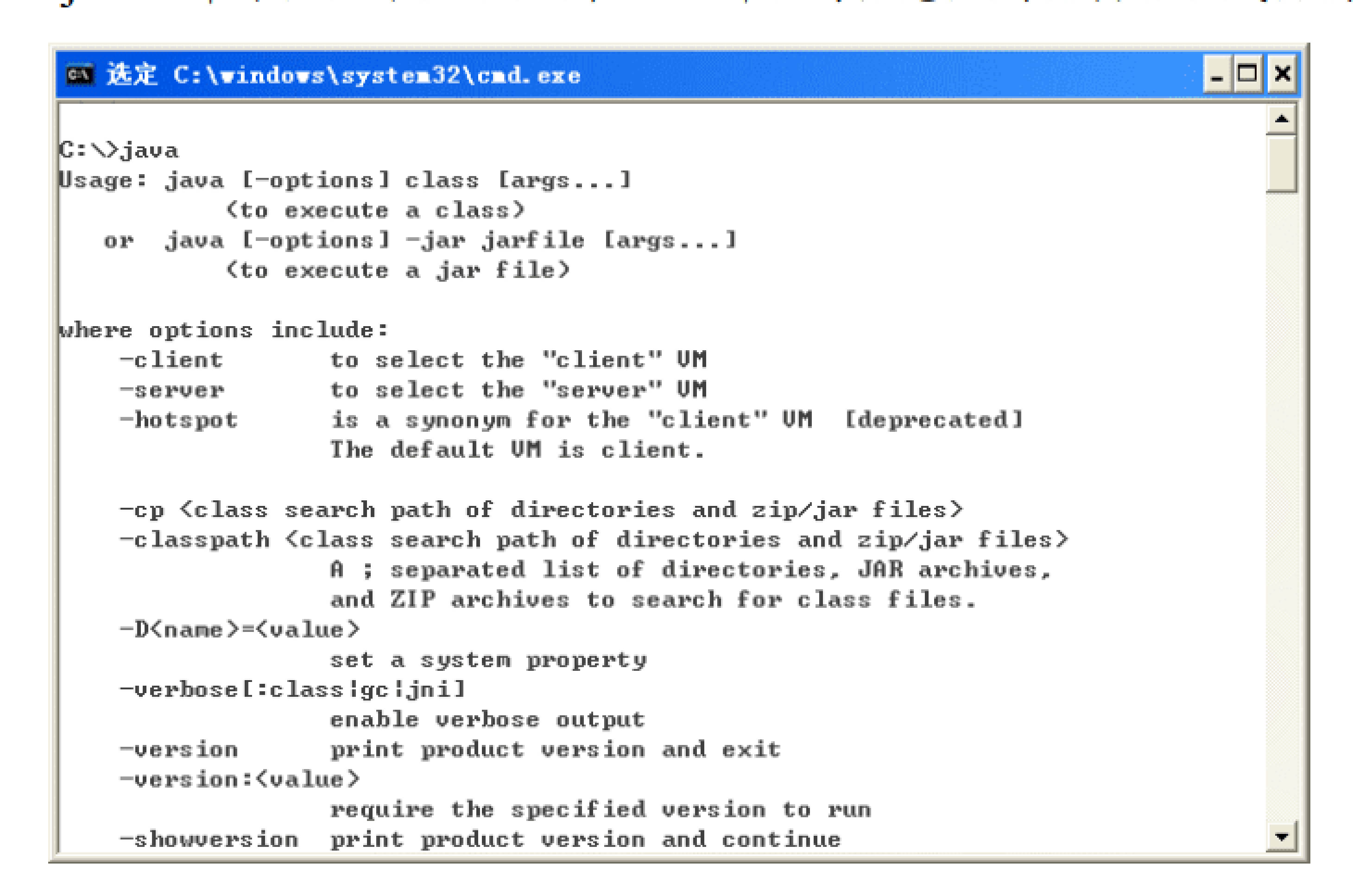


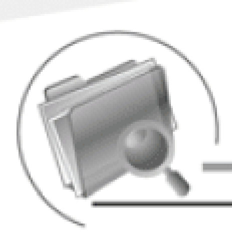
图 2-11　测试“java”命令

2.1.4　Tomcat 的安装与配置

Tomcat 是 Apache 开源组织 Jakarta 项目下的一个产品，它支持 JSP 2.0 和 Servlet 2.4 规范，是一个免费的 J2EE 应用服务器。当前的最新版本是 5.5.9，可以在 Apache 的 Jakarta 项目网站上找到。在 http://jakarta.apache.org/tomcat/index.html 中即可找到安装程序的下载地址。下面简单介绍如何安装和配置 Tomcat。

(1) 打开 Tomcat 安装程序所在文件夹，双击 jakarta-tomcat-5.5.9.exe 文件执行安装程序。弹出如图 2-12 所示的欢迎对话框。在安装 Tomcat 时最好关闭其他所有应用程序，这样可以保证不必重启计算机就能完成安装。



(2) 单击【Next】(下一步)按钮，弹出许可证协议(License Agreement)对话框，如图 2-13 所示。

(3) 单击【I Agree】(同意)按钮，进入自定义安装(Choose Components)对话框，如图 2-14 所示。在该对话框中，可以选择 4 种安装类型中的一种：Normal(正常安装，不包括实例，不使用 Windows 服务)、Minimum(最小安装，不包括文档、实例和服务)、Full(完全安装)和 Custom(自定义安装)。如果直接选择要安装的可选组件，则默认为 Custom 类型。所有组件(Component)中只有 Tomcat Core(内核)是必选的，可选组件包括：Tomcat – Service(使用 Windows 服务在开机的时候自动启动 Tomcat 服务，只适用于 Windows NT 4.0、2000 和 XP 系统)、Start Menu Items(在【开始】菜单中加入相应的快捷方式)、Documentation(文档)、Examples(实例)和 Webapps(用于部署 J2EE 应用的目录)。从图 2-14 中可以看到，即使选择了所有组件，Tomcat 也只占用 11.7MB 的磁盘空间，和 WebSphere、WebLogic 等重量级服务器相差甚远。

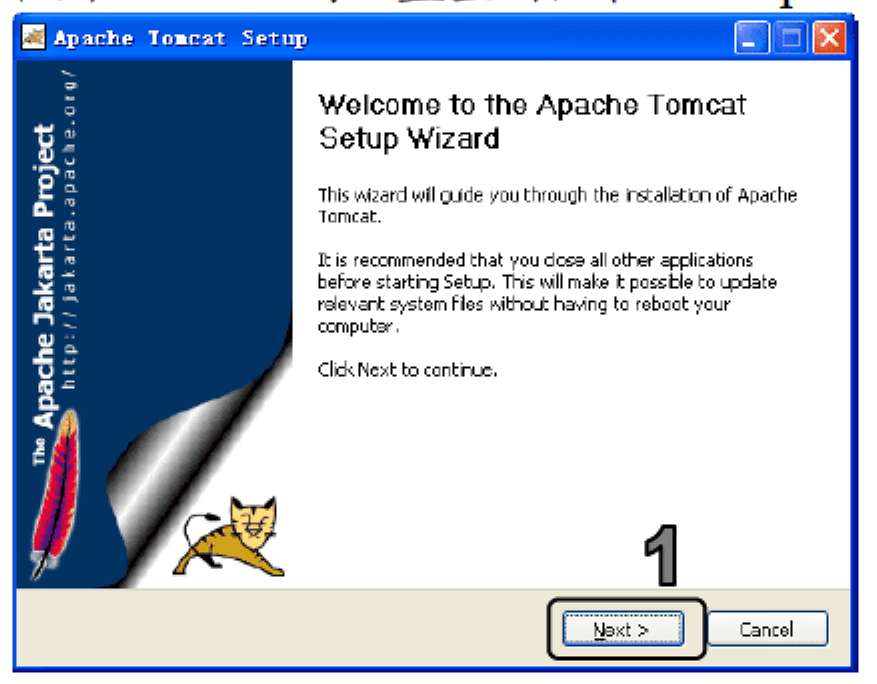


图 2-12　Tomcat 安装程序的欢迎对话框

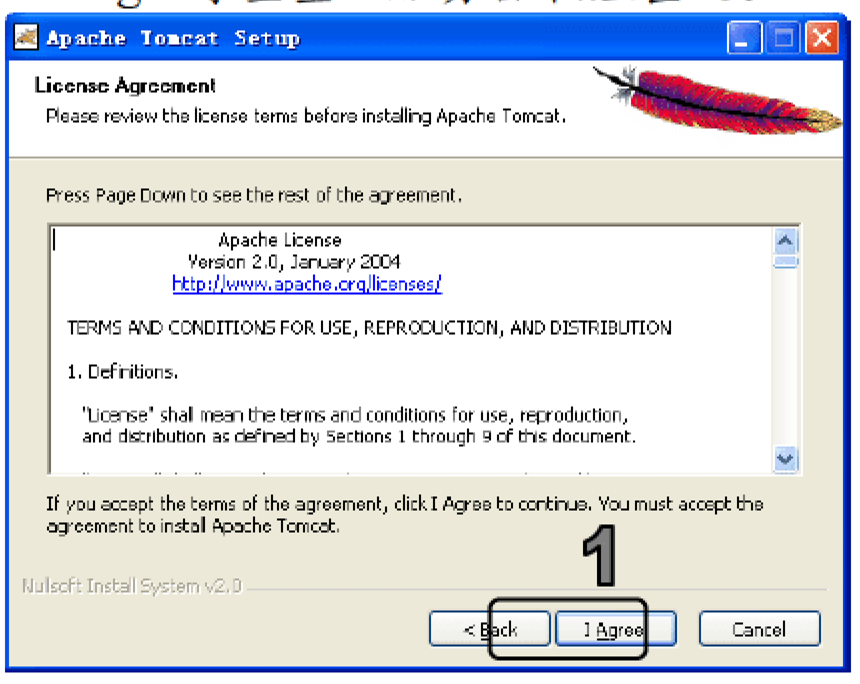


图 2-13　Tomcat 许可证协议对话框

(4) 选择【Full】(完全安装)选项，单击【Next】(下一步)按钮，弹出选择安装路径(Choose Install Location)对话框，如图 2-15 所示。单击【Browse】(浏览)按钮选择安装目标文件夹或在文本框中直接输入目标文件夹。

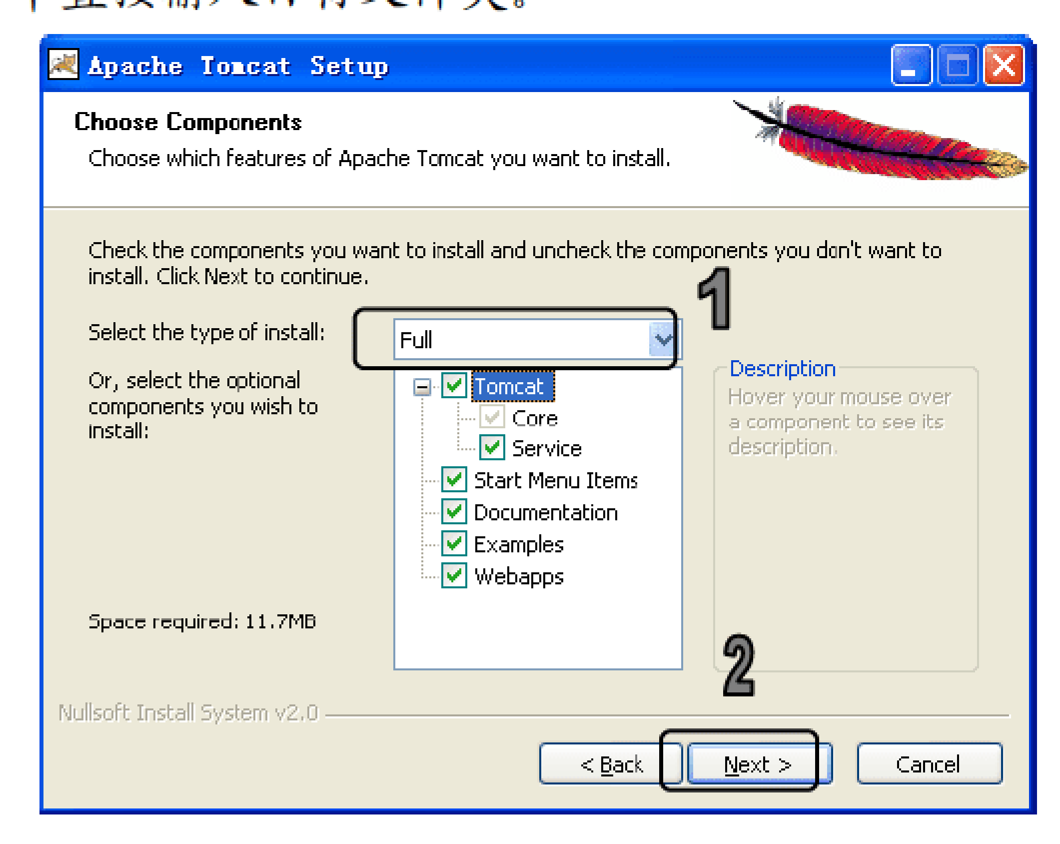


图 2-14　Tomcat 自定义安装对话框

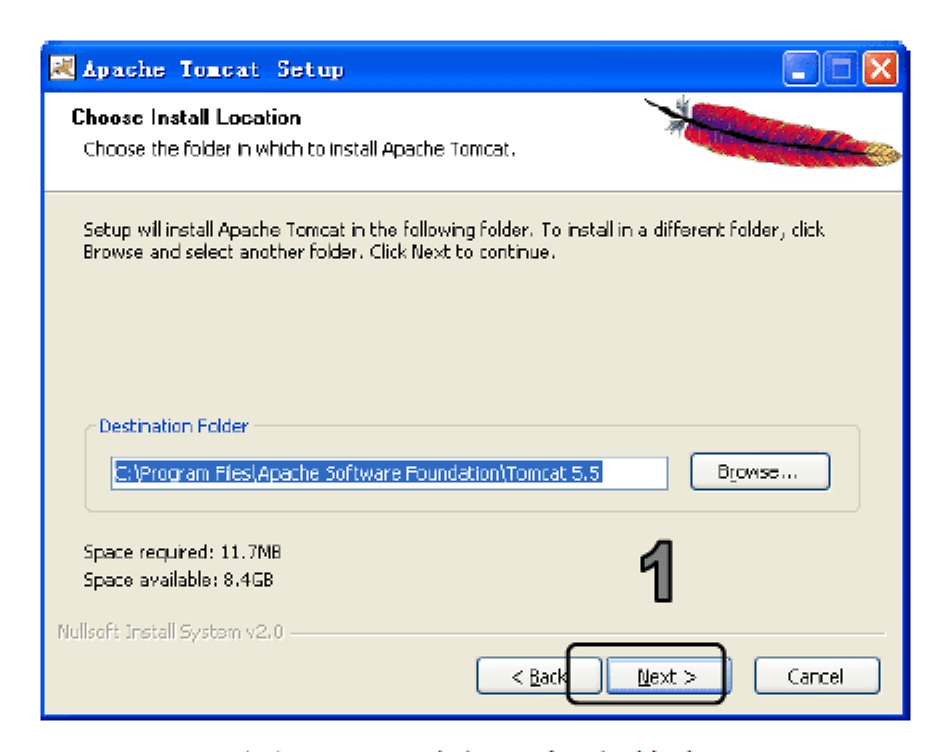


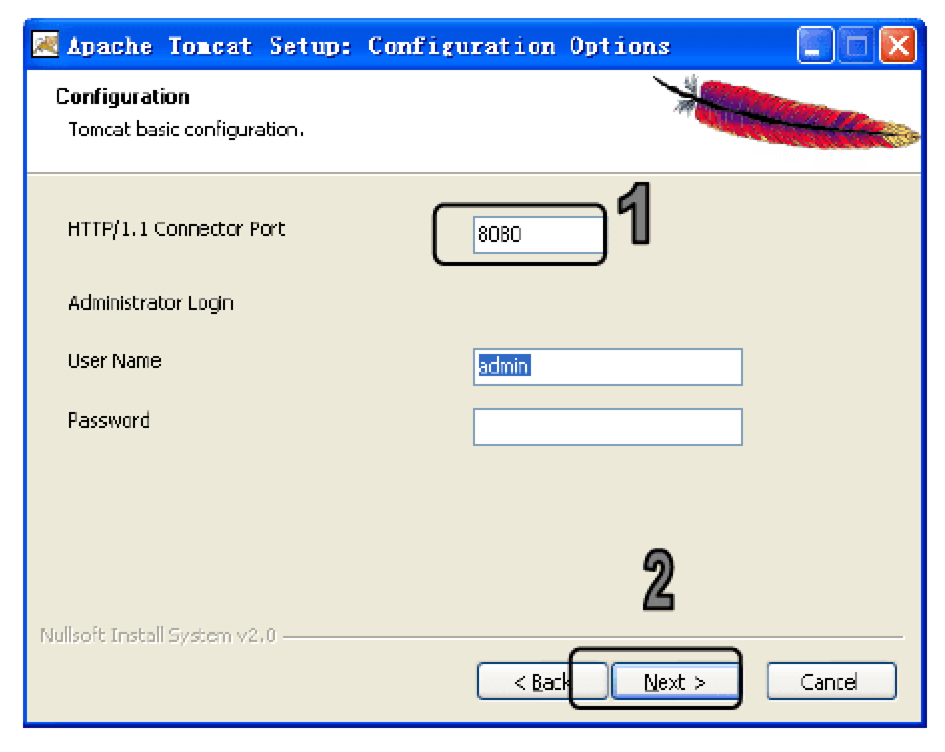
图 2-15　选择目标文件夹

(5) 单击【Next】(下一步)按钮，弹出如图 2-16 所示的 Configuration(配置)对话框。在该对话框中可以配置 3 个选项：HTTP/1.1 Connection Port 用来确定 Tomcat 服务器监听的连接端口，



默认为 8080，如果在本机没有其他 Web 服务器，也可以将其设置为 80 端口；Administrator Login(系统管理员登录账号)的 User Name 是系统管理员登录进行管理时使用的用户名，默认为 Admin；Password 是其使用的密码，可以设置为空。

(6) 单击【Next】(下一步)按钮，出现如图 2-17 所示的确认 JRE 1.5.0 虚拟机(Java Virtual Machine)安装路径的对话框弹出。如果已经成功安装了 JRE 1.5.0，则其安装路径会自动出现在文本框中。如果不能自动出现，可以手工输入或通过浏览按钮指定路径。如果没有安装相应的 JRE 1.5.0 则需要单击【Cancel】(取消)按钮退出安装程序，如 2.1.3 节的介绍先安装完 JDK/JRE 后重新安装 Tomcat。如果强行继续安装，则在运行 JSP 应用程序时会出错。





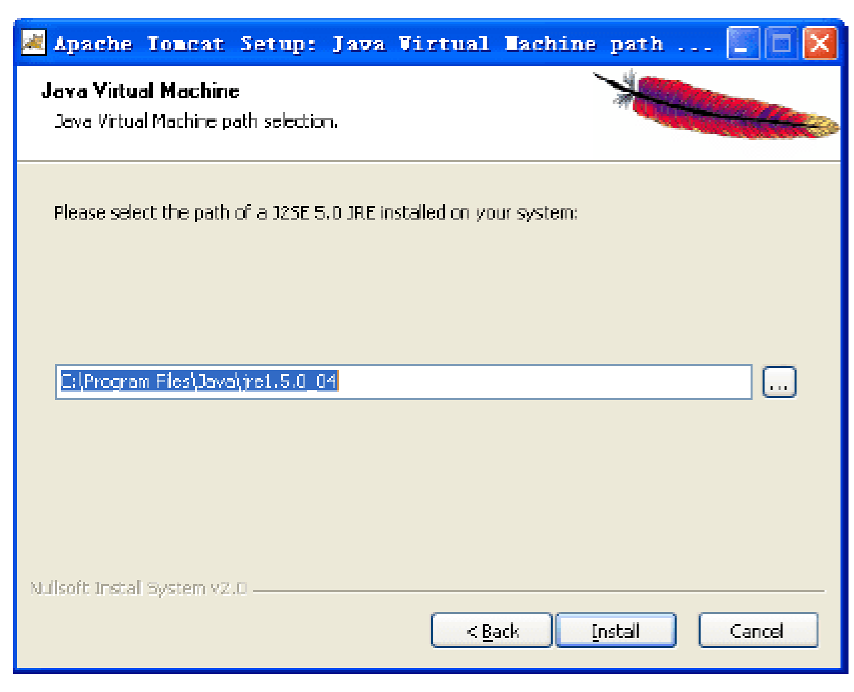
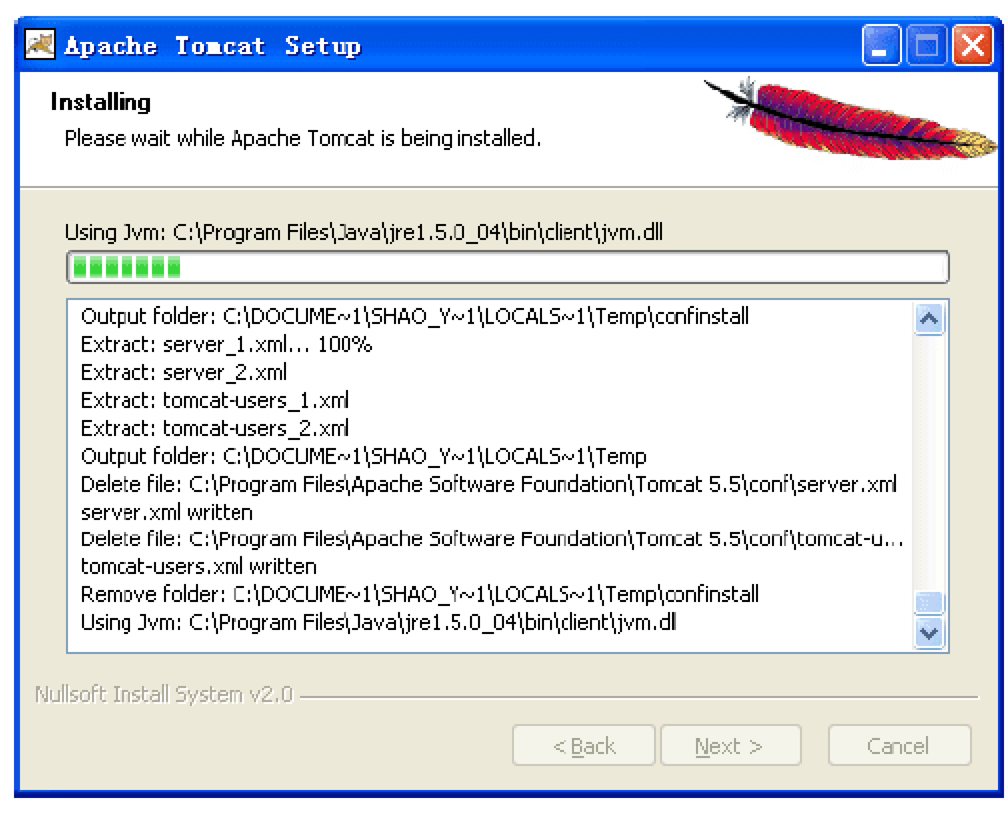


图 2-16　配置对话框　　　　图 2-17　Java 虚拟机路径



(7) 单击【Install】(安装)按钮即可开始安装过程，如图 2-18 所示。

(8) 安装结束后会出现如图 2-19 所示的对话框，提示安装完成。选中复选框【Run Apache Tomcat】则开始运行 Tomcat，而选中复选框【Show Readme】则显示自述文件。



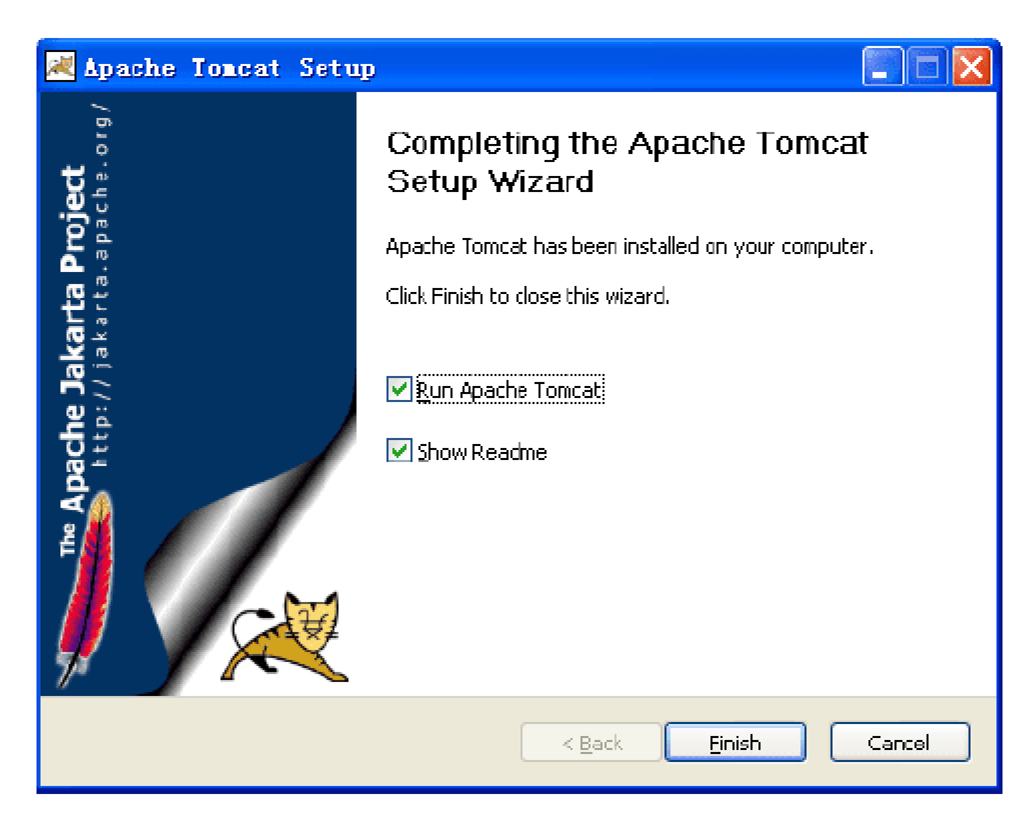


图 2-18　安装进度　　　　图 2-19　安装完成

(9) 这里只选中【Run Apache Tomcat】复选框，单击【Finish】(完成)按钮关闭安装向导，同时启动 Tomcat 服务，如图 2-20 所示。

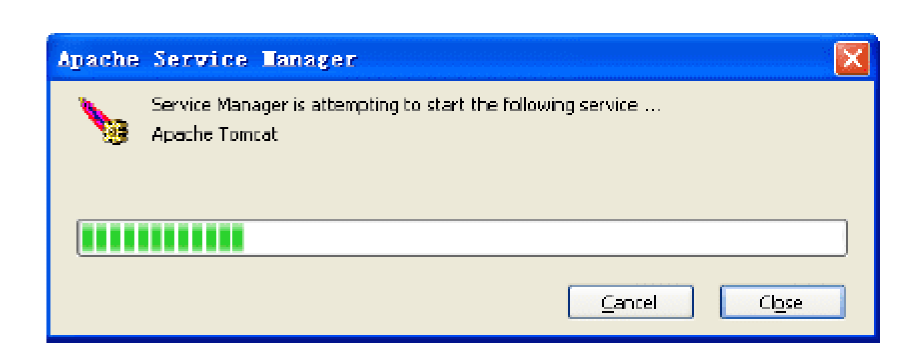


图 2-20　启动 Tomcat 服务

(10) 启动完成后，在屏幕右下角的 Windows 任务栏上将出现一个图标标志显示 Tomcat 服务正在运行。打开 IE 浏览器，在地址栏中输入 http://localhost:8080，如果出现如图 2-21 所示的页面，则表明 Web 服务器已经正常运行。通过页面的 JSP Examples 链接打开一个 JSP 示例，或者在页面地址中直接输入 http://localhost:8080/jsp-examples/jsp2/el/basic-arithmetic.jsp(这是 Tomcat 自带的示例之一)，如果出现如图 2-22 所示的页面，则表明 JSP 引擎已经就绪，可以在服务器上部署 JSP 应用了。

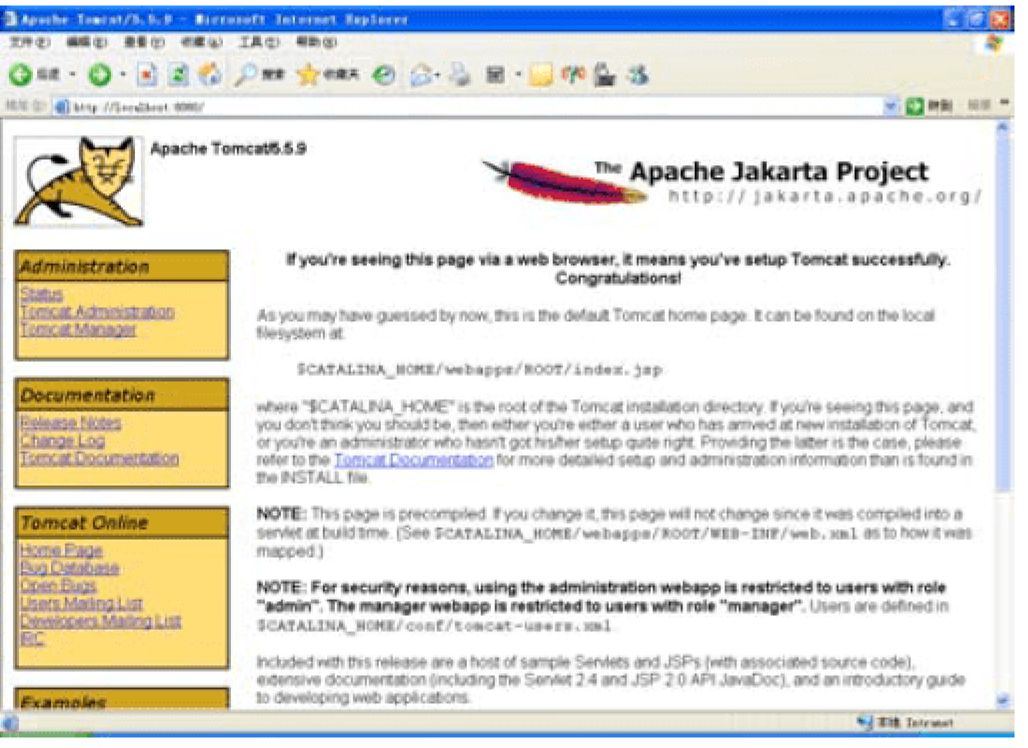


图 2-21　Tomcat 服务器首页

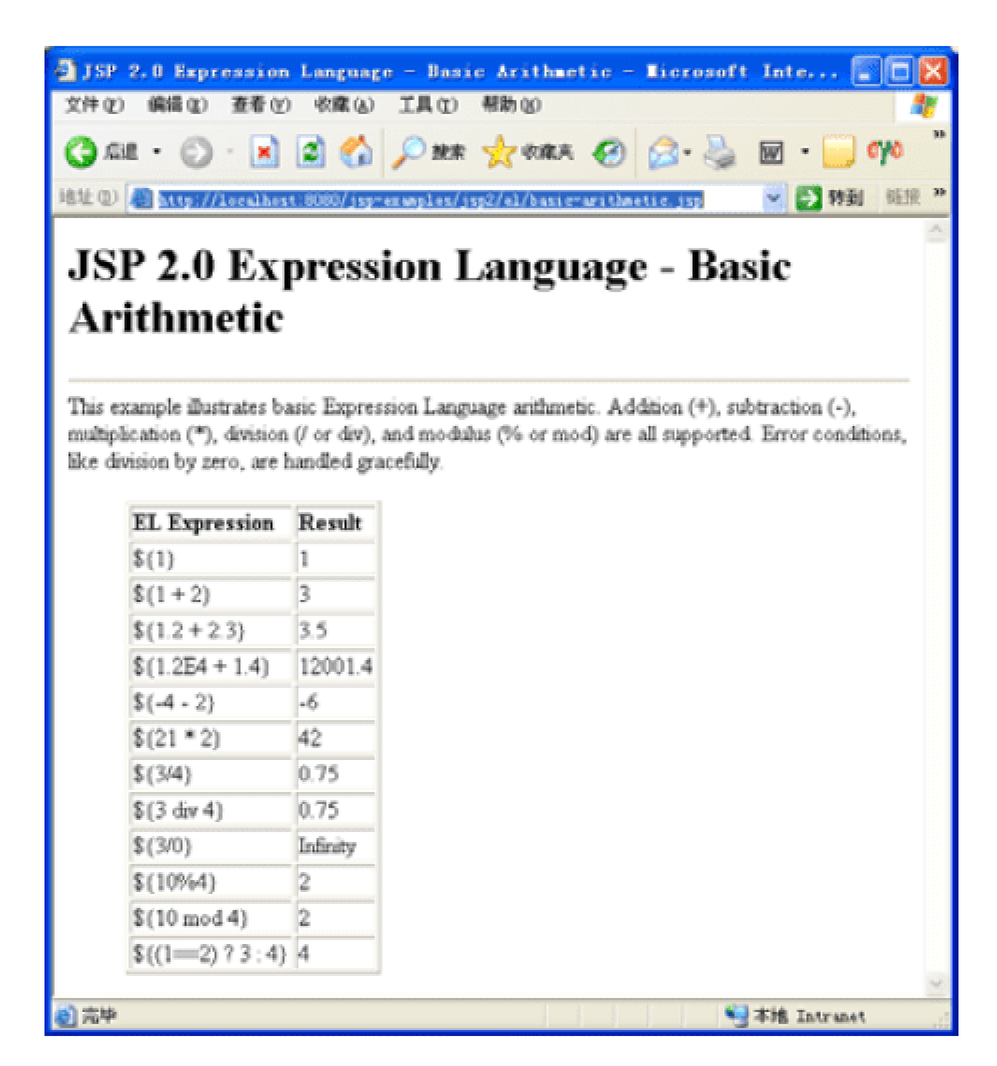


图 2-22　JSP 示例



2.2　开发环境

2.2.1　JSP 开发环境

JSP 的开发环境与 JSP 服务器端的运行环境一样，需要有 Java 开发包(JDK)，Web 服务器和包含 JSP、Servlet 容器的 J2EE 应用服务器。在本书中将以 JDK 1.5.0 和 Tomcat 5.5.9 作为开发环境。

由于 JSP 是在 HTML 中内嵌 Java 代码，所以使用一般的文本编辑器就可以进行 JSP 应用的开发。但是，为了更直观的设计和更简便的调试，还是需要有比较好的集成工具进行开发。JBuilder 和 WSAD 是两个功能非常强大的 Java 开发工具，但这两个重量级的庞然大物对于资源的需求非常大，对于初学者和开发普通 Web 商务应用来说有些大材小用。所以建议读者采用开源的 Eclipse 作为主要的开发工具，配合比较流行的网页制作工具，如 Dreamweaver，一定会有事半功倍的效果。



2.2.2　Eclipse 的安装

Eclipse 是一个非常强大的集成开发工具，事实上，它不仅仅用于开发 Java 应用，而且可以用来开发 C/C++、COBOL 等语言的应用。Eclipse 同时也是负责开发 Eclipse 工具的开源项目的名字。

Eclipse 开发工具采用插件技术，它的核心部分提供了一个具有丰富特性的开发环境，本身并不提供大量的最终用户功能，而是通过插件来快速开发集成功能部件。其核心是动态发现插件的体系结构。Eclipse 负责处理基本环境的后台工作，并提供标准的用户导航模型，而每个插件则专注于执行少量的任务。通过集成大量的插件，Eclipse 的功能可以不断扩展，以支持不同的应用。目前，Eclipse 组织和其他第三方供应商都提供了众多的插件来管理多种开发任务，其中包括测试、性能调整和程序调试等。总而言之，Eclipse 平台是一个成熟的、精心设计的、可扩展的体系结构，开发人员使用起来非常方便。

Eclipse 是一个完全免费的软件，可以到其官方网站 http://www.eclipse.org 免费下载。在这个网站上可以找到 Eclipse 平台以及其他用于开发 JSP 应用非常有用的插件。本书用到的某些 Eclipse 插件可能来自于第三方供应商，在提及到这些插件时会做相应介绍。目前 Eclipse 的最新稳定版本是 3.1 版，本书将使用 Eclipse 3.1 进行介绍。下面将简单介绍如何安装 Eclipse 以及如何安装配置 Lomboz 和 Sysdeo Tomcat 插件以开发 JSP 应用。

(1) 从官方网站下载 Eclipse 安装程序 eclipse-SDK-3.1-win32.zip。将其复制到安装目标文件夹下，如 E:\。使用 WinZip 或 WinRAR 等解压缩工具将其解压缩到该目录下。解压缩后的

所有文件都位于 E:\eclipse 目录下，它的子目录结构如图 2-23 所示。

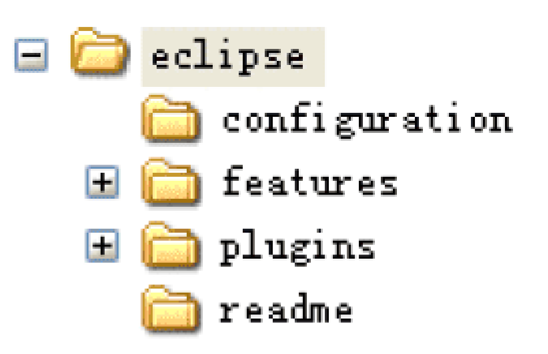


图 2-23　eclipse 子目录结构

(2) 将语言包全部解压之后，打开 eclipse 目录，双击 eclipse.exe 文件打开 Eclipse 程序。如图 2-24 所示的带有半月形 Eclipse 图标的欢迎界面将出现在屏幕上。

(3) 初次启动时，会弹出如图 2-25 所示的 Workspace Launcher(工作空间启动程序)对话框，Eclipse 将项目存储在一个称为工作空间(Workspace)的目录中，可以使用默认的 workspace 作为工作空间目录或者使用 Browse 按钮指定其他目录。如果希望未来的项目都放在这个空间中，可以选中【Use this as the default and do not ask again】(将此值用作默认值并且不再询问)复选框，然后单击【OK】按钮。

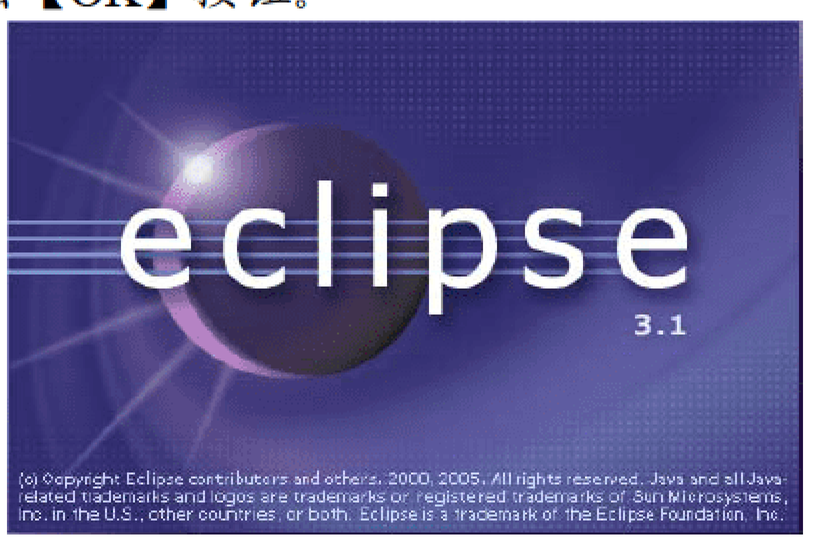


图 2-24　eclipse 欢迎页面

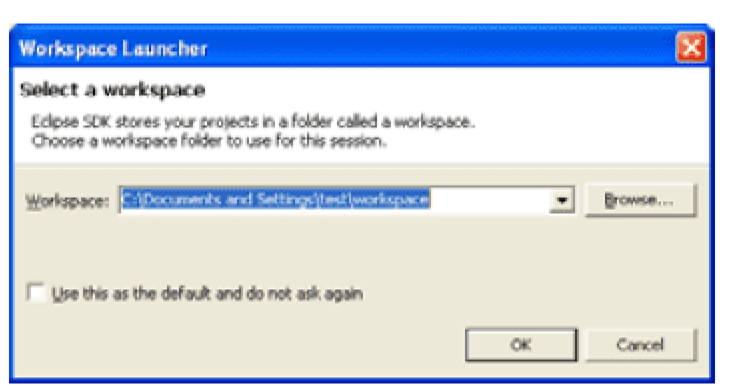


图 2-25　Workspace Launcher 对话框

(4) Eclipse 平台的主界面如图 2-26 所示，接下来就可以使用 Eclipse 开发应用程序了。

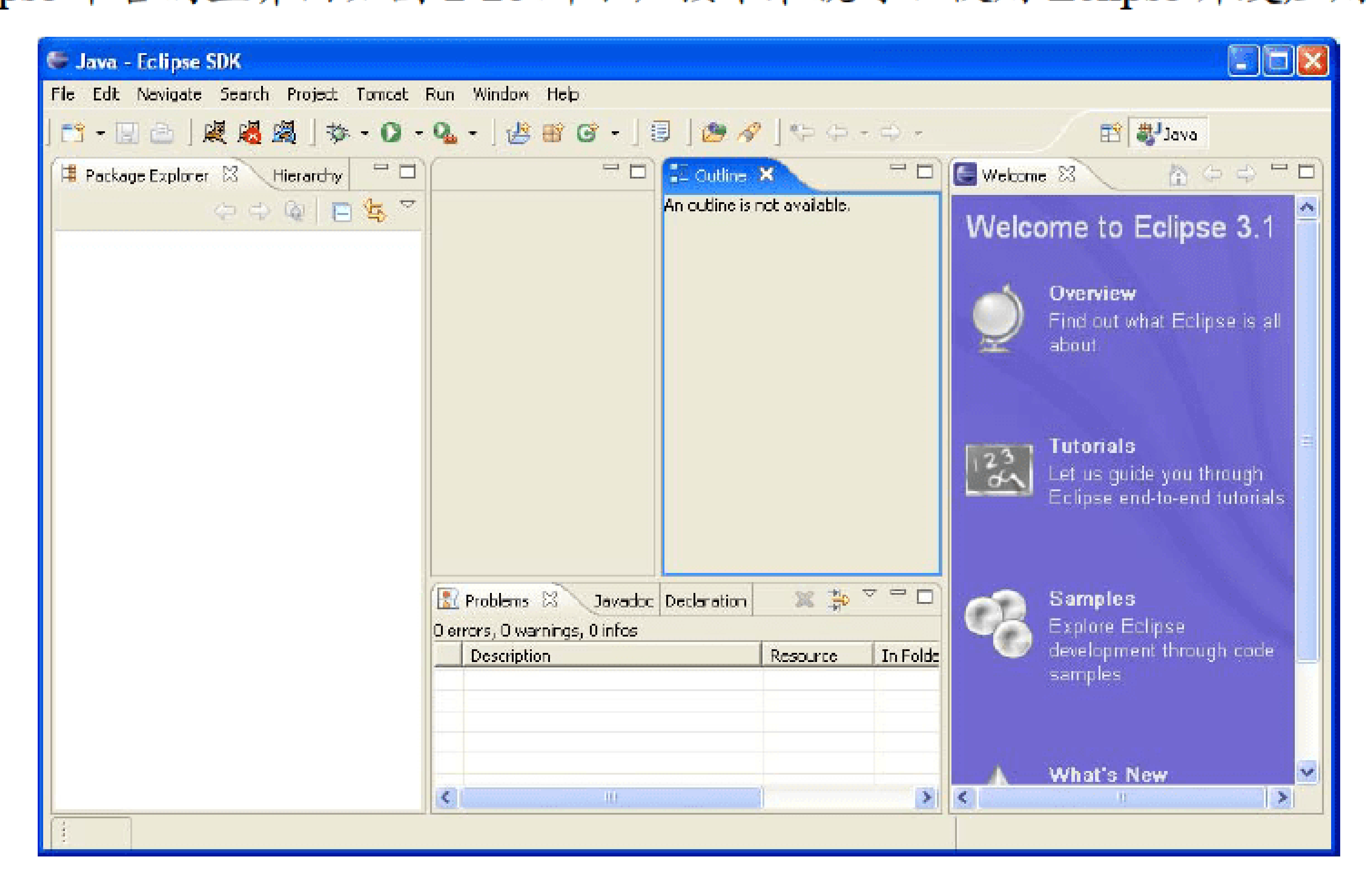


图 2-26　Eclipse 应用主界面



(5) 为了方便使用 Eclipse 开发 JSP 应用，还需要安装一些插件。这里介绍两个比较流行的插件：Lomboz 和 Sysdeo Tomcat。Lomboz 是 ObjectWeb 开发的一个用于开发 J2EE 应用的 Eclipse 平台插件，它提供了很好的代码辅助输入和代码调试功能。该插件可以在 http://www.objectlearn.com/index.jsp 网站上下载。需要注意的是，Lomboz 插件的安装需要 Eclipse 安装 GEF、JEM 和 EMF 等几个插件。在网站上也提供了包括这些插件的完整安装程序的下载 lomboz-eclipse-emf-gef-jem-3.1M6.zip。将此文件放到 eclipse 父目录中并解压缩即可。

(6) SysdeoTomcat 插件能够将 Eclipse 平台和 Tomcat 有机地结合起来，可以在 Eclipse 平台内部控制 Tomcat 的启动、停止和 Web 应用的部署。对于处于开发期间的 JSP 应用来说这是一个很好的工具，它避免了开发人员不断地在 Tomcat 服务器上手工的停止、部署和重新启动 Tomcat 服务。Tomcat 插件可以在 http://www.sysdeo.com/eclipse/tomcatplugin 网站上找到。下载该插件后，将其解压缩到 eclipse 下的 plugins 子目录中即可。

(7) 重新启动 Eclipse，会发现 Eclipse 主菜单中多了一个 Tomcat 菜单项，说明 Tomcat 插件已经正确安装。这里还需要进行一些配置才能将 Eclipse 与 Tomcat 结合起来。选择【Window】|【Preferences】菜单，打开【Preferences】对话框。选中左侧列表中的 Tomcat 选项，如图 2-27 所示。在【Tomcat version】选项组中选中【Version 5.x】单选按钮，在【Tomcat home】文本框中输入 Tomcat 所安装的路径，在【Context declaration mode】选项组中选中 Server.xml 单选按钮，并将【Configuration file】选项设置为“Tomcat 目录/conf/server.xml”。



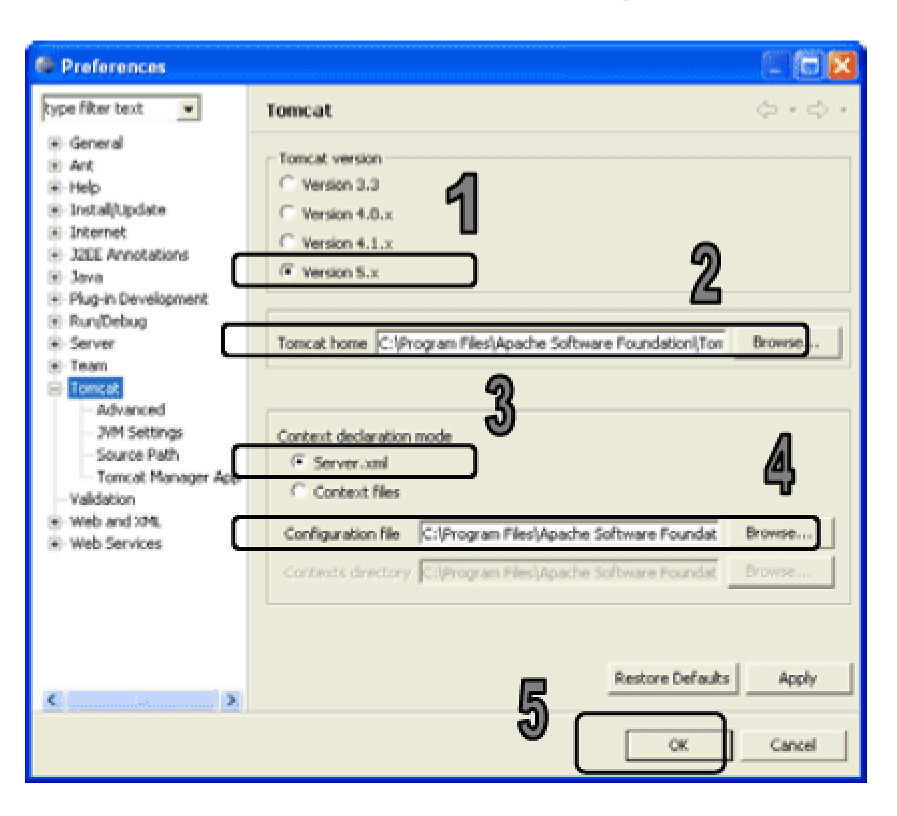


图 2-27　Tomcat 插件配置页

(8) 单击展开左侧的【Tomcat】选项，选择【Advanced】选项，在右侧的【Tomcat base】文本框中输入 Tomcat 的安装地址，如图 2-28 所示。单击【OK】按钮结束配置，接下来就可以使用 Eclipse 来控制 Tomcat 的启动、终止和调试了。

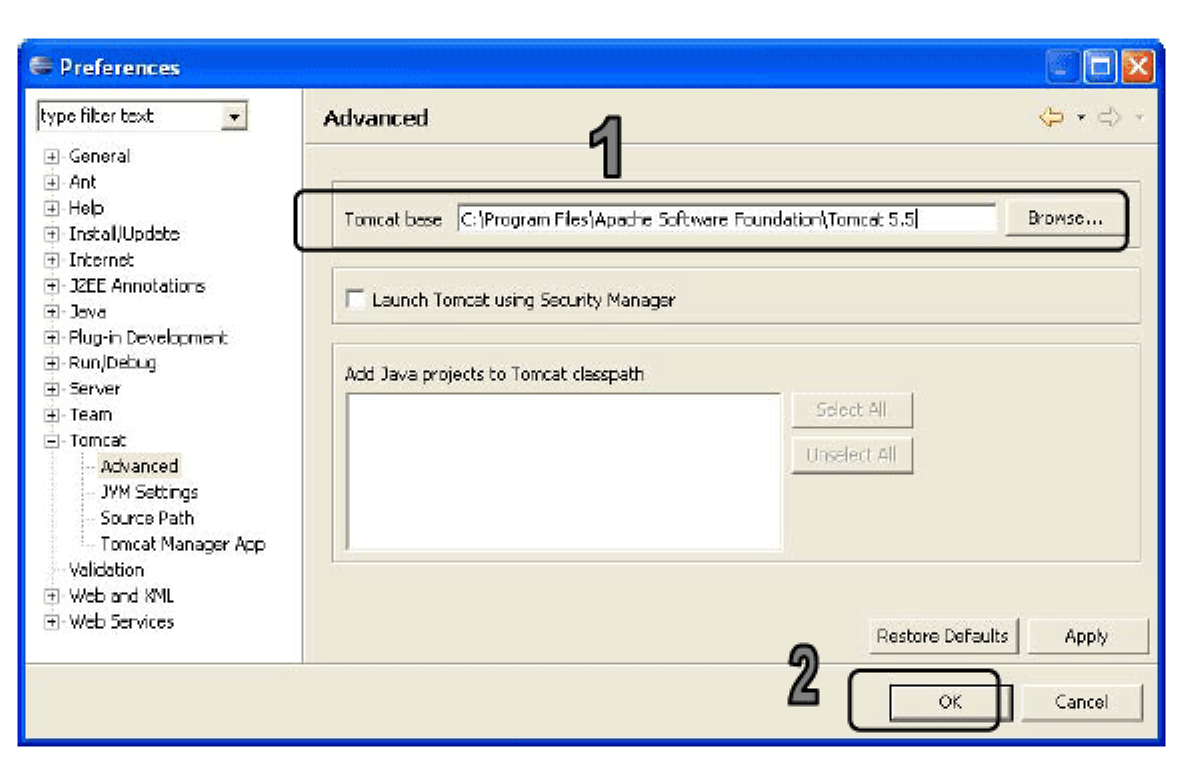


图 2-28 Tomcat Advanced 配置页



2.2.3 开发方式

JSP 作为 J2EE 的一部分，既可以用于开发小型的 Web 站点，也可以用于开发大型的、企业级的应用程序。根据开发的目标程序不同，使用的开发方式也不同。JSP 开发 Web 站点主要有以下几种方式。

⦿ 单独使用 JSP

对于小型的 Web 站点，可以单独使用 JSP 来构建动态网页，这种站点最为简单，所需要的仅仅是简单的数据库查询、动态日期等基本的功能。对于这种开发模式，一般可以将所有的动态处理部分都放置在 JSP 的 Scriptlet 中，就像一般使用 PHP 或 ASP 开发动态网页一样。

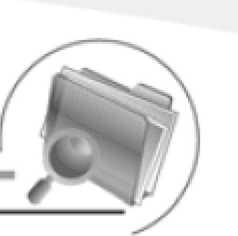
⦿ JSP+JavaBean

中型站点面对的是一定量的数据库查询、用户管理和小量的商业业务逻辑。对于这种站点，不能将所有的东西全部交给 JSP 页面来处理。在单纯的 JSP 中加入 JavaBean 技术将有助于这种中型站点的开发。利用 JavaBean，将很容易完成诸如数据库连接、用户登录与注销、商业业务逻辑封装的任务。例如：将常用的数据库连接写为一个 JavaBean，既方便了使用，又可以使 JSP 文件简单而清晰。通过封装，还可以防止一般的开发人员直接获得数据库的控制权。在本书第 15 章中作者采用的就是这种开发方式来构建 Web 动态教务系统。

⦿ JSP+JavaBean+Servlet

无论用 ASP 还是 PHP 开发动态网站，长期以来都有一个比较重要的问题，就是网站的逻辑关系和网站的显示页面不容易分开。常常可以看见一些夹杂着 if…then、case…select 或是 if{…}和大量显示用的 HTML 代码的 ASP、PHP 页面。即使是有着良好的程序写作习惯的程序员，其程序也几乎无法阅读。另一方面，动态 Web 的开发人员也在抱怨，将网站美工设计的静态页面和动态程序和并的过程是一个异常繁琐的过程。

事实上，在逻辑关系异常复杂的网站中，借助于 Servlet 和 JSP 良好的交互关系和 JavaBean 的协助，完全可以将网站的整个逻辑结构放在 Servlet 中，而将动态页面的输出放在 JSP 页面中



来完成。在这种开发方式中，一个网站可以有一个或几个核心的 Servlet 来处理网站的逻辑，通过调用 JSP 页面来完成客户端(通常是 Web 浏览器)的请求。在 J2EE 模型中，Servlet 的这项功能已被 EJB 取代。

◉ J2EE 开发模型

在 J2EE 开发模型中，整个系统可以分为 3 个主要的部分：视图、控制器和模型。

视图就是用户界面部分，在 Web 应用程序中也就是 HTML、XML 及 JSP 页面。这个部分主要处理用户看到的内容，动态的 JSP 部分处理了用户可以看见的动态网页，而静态的网页则由 HTML、XML 输出。

控制器负责网站的整个逻辑，用于管理用户与视图发生的交互。可以将控制器想象成处在视图和数据之间，对视图如何与模型交互进行管理。通过使视图完全独立于控制器和模型，就可以轻松替换前端客户程序。也就是说，网页制作人员将可以独立自由地改变 Web 页面而不用担心影响这个基于 Web 的应用程序的功能。在 J2EE 中，控制器的功能一般是由 Servlet、JavaBean、Enterprise JavaBean 中的 Session Bean 来担当的。

模型就是应用业务逻辑部分，这一部分的对应模块就是 Enterprise JavaBean。借助于 EJB 强大的组件技术和企业级的管理控制，开发人员可以轻松地开发出可供复用的业务逻辑模块。



2.3　上机练习

本章上机实验主要以安装和配置 JSP 的运行环境和开发环境所需的部分软件为主。其中，读者应该重点掌握安装 JDK、安装和配置 Tomcat 以及安装和配置 Eclipse 开发工具。

下面以安装 Eclipse 为例，进行练习。

(1) 从 Eclipse 官方网站(www.eclipse.org)中搜索 Eclipse SDK 3.1。

(2) 在相应的网页上单击下载，将 eclipse-SDK-3.1-win32.zip 下载到安装目录，如 E:\。

(3) 打开【我的电脑】，找到 E 驱动器下的 eclipse-SDK-3.1-win32.zip 文件。

(4) 右键单击 eclipse-SDK-3.1-win32.zip 文件，(假设已经安装 WinRAR 中文版)在弹出的快捷菜单中选择【解压到当前文件夹】命令。

(5) 打开解压缩后的 eclipse 文件夹，双击 eclipse.exe 文件启动 Eclipse。

(6) 在【Workspace Launcher】对话框中指定工作空间位置，单击【OK】按钮即可运行 Eclipse。

2.4　习题

2.4.1　填空题

1. 运行 JSP 服务器端需要安装和配置____________、____________

和___________________。

2. Eclipse 利用_______________来实现功能的扩展。

2.4.2 选择题

1. 在以下操作系统中，支持 Java/JSP 的有哪些？(　　　　) (多选)

A. Windows　　B. Solaris　　C. UNIX　　D. Linux

2. 以下选项中(　　　　)不是开发 JSP 应用程序所必需的。

A. JDK　　B. J2EE SDK

C. 应用服务器　　D. 开发工具 Eclipse

2.4.3 问答题

简要描述 JSP Web 应用的客户端和服务器端的运行环境。



JSP 语 法

学习目标

JSP 技术采用在 HTML 代码中内嵌 Java 代码的方式实现静态和动态内容的结合。因此，学好 JSP 就需要了解 HTML 和 Java 语言，同时还需要理解 JSP 如何将 HTML 和 Java 结合起来，使服务器能够方便快捷地加以解释。

本章主要介绍 JSP 的基本语法，解释其基本功能和作用。首先给出 JSP 的概况、工作原理，然后介绍 JSP 的各种语法和语义。

本章重点

- JSP 容器
- JSP 的基本结构
- JSP 的语法
- JSP 指令
- JSP 操作

3.1 JSP 概述

3.1.1 JSP 容器

Java2 企业版(J2EE)定义了几个容器，包括 JSP 容器、服务器小程序和企业级 JavaBeans 容器。容器为企业组件提供了在其中生存和活动的总体运行时环境，它管理组件的生存期并向组件提供不同的服务。此外，它还协调组件与更大的运行时环境之间的交互。

每一个 J2EE 容器都为它所担负责任的组件提供服务。JSP 容器将 JSP 转换为 Java 服务器小程序(Servlet)代码，然后将结果编译并加载到服务器小程序容器中。另外，它还协调服务器小程序容器与编译过的 JSP 之间的关系。服务器小程序容器为 Java 服务器小程序提供运行时环境。

3.1.2 JSP 页面

JSP 页面不会以其本来面目显示在网页上，JSP 容器会把 JSP 页面转换为 Servlet 代码，并重新解释。当浏览器第一次请求 JSP 页面时，将依次发生以下事件：

(1) 解释 JSP 页面。

(2) 生成 Java 服务器小程序(Servlet)代码。

(3) 使用与 JSP 容器打包在一起的标准 Java 编译器将生成的服务器小程序编译为 Java 字节码。

(4) 将服务器小程序加载到服务器小程序容器的 Java 虚拟机(Java Virtual Machine JVM)中。

(5) 调用服务器小程序的 service 方法。

如果浏览器以后请求相同的 JSP 页面，那么只需要执行上面的步骤(5)即可。但是当 JSP 页面发生变化时，必须重复上面 5 个步骤。从生成的文件代码来看上述步骤可以简化为如图 3-1 所示的过程。

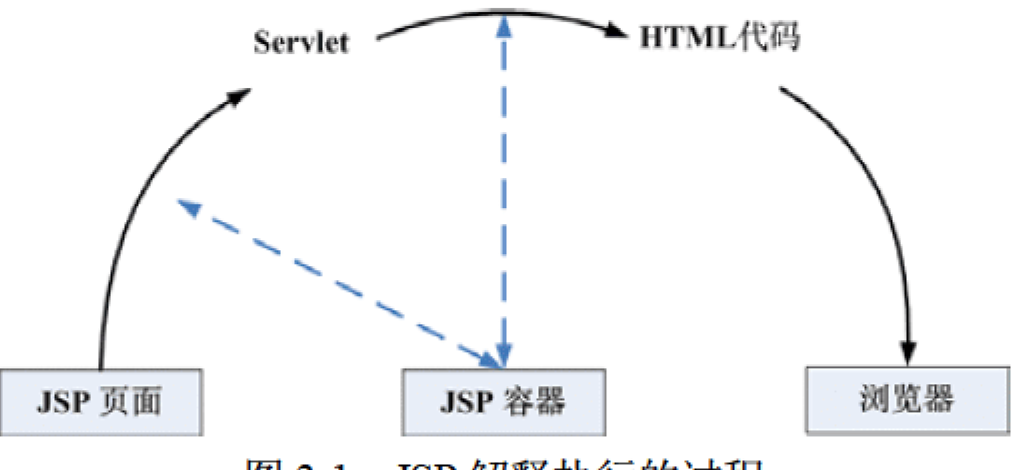


图 3-1　JSP 解释执行的过程

下面看一个非常简单的 JSP 页面 testJSP.jsp，它的功能是在页面上显示“简单的 JSP 页面”几个字，代码如下。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page language="java" contentType="text/html; charset=GB2312" %>
<%@ page session="false"%>
<html>
  <head>
    <% String str="简单的 JSP 页面"; %>
  </head>
  <%=str%>
</html>
```

将该 JSP 文件放置到通过 Eclipse 创建的 Tomcat 项目目录下(创建过程将在后面章节中介



绍)，启动 Tomcat 服务器。在客户端浏览器中输入该 JSP 页面对应的 URL，第一次浏览时会经过短暂的时间响应，最后显示“简单的 JSP 页面”。同时，在该项目文件夹 work/org/apache/jsp 子目录下可以找到一个名为 testJSP_jsp.java 的文件，用文本编辑器打开该文件，可以看到如下转译后的 Servlet 源代码：

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class testJSP_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
  private static java.util.Vector _jspx_dependants;
  public java.util.List getDependants() {
    return _jspx_dependants;
  }
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html; charset=GB2312");
      pageContext = _jspxFactory.getPageContext(this, request, response,
         null, false, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      out = pageContext.getOut();
      _jspx_out = out;
      out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">\r\n");
      out.write("\r\n");
      out.write("\r\n");
      out.write("<html>\r\n");
      out.write("   <head>\r\n");
```



```
            out.write("        ");
        String str="简单的 JSP 页面";
            out.write("\r\n");
            out.write("    </head>\r\n");
            out.write("    ");
            out.print(str);
            out.write("\r\n");
            out.write("</html>");
          } catch (Throwable t) {
            if (!(t instanceof SkipPageException)){
              out = _jspx_out;
              if (out != null && out.getBufferSize() != 0)
                out.clearBuffer();
              if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
            }
          } finally {
            if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
          }
        }
      }
```

对生成的 Servlet 各个模块的解释可参见本书第 8 章中的相关介绍。

3.1.3 JSP 的作用域

JSP 页面的生成是从一个请求开始的，但是它所创建的一些 Java 对象的生命周期却可以跨越多个请求。所有的这些对象都有一个 scope 属性，它定义了指向该对象的引用在何处可用以及容器何时删除该对象。

JSP 容器支持 4 种不同的作用域：

⊙ 页面

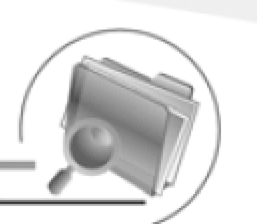
只能在创建对象的 JSP 页面内部使用【页面作用域】引用这些对象。JSP 容器在 JSP 页面返回一个响应或是将请求转发给另一个页面之后将删除所有这些对象。

⊙ 请求

可以使用【请求作用域】从处理同一请求的任何页面访问对象。一个页面可能将请求转发给另一个页面，这样就会有多个页面处理同一个请求。所有的这些页面都可以访问请求作用域内的对象，JSP 容器将在请求完成后删除这些对象。

⊙ 会话

可以使用【会话作用域】使同一个会话中所有页面都能访问这些对象。JSP 容器将在会话



结束时删除这些对象。

◉ 应用程序

可以使用【应用程序作用域】在同一个 Web 应用程序内的任何位置访问这些对象。JSP 容器将在重新加载服务器小程序环境时(通常在服务器重新启动时)删除这些对象。

3.1.4　JSP 的结构

按类别组织的 JSP 结构图表可以帮助用户更清晰地了解它，如图 3-2 所示为 JSP 的组织结构。

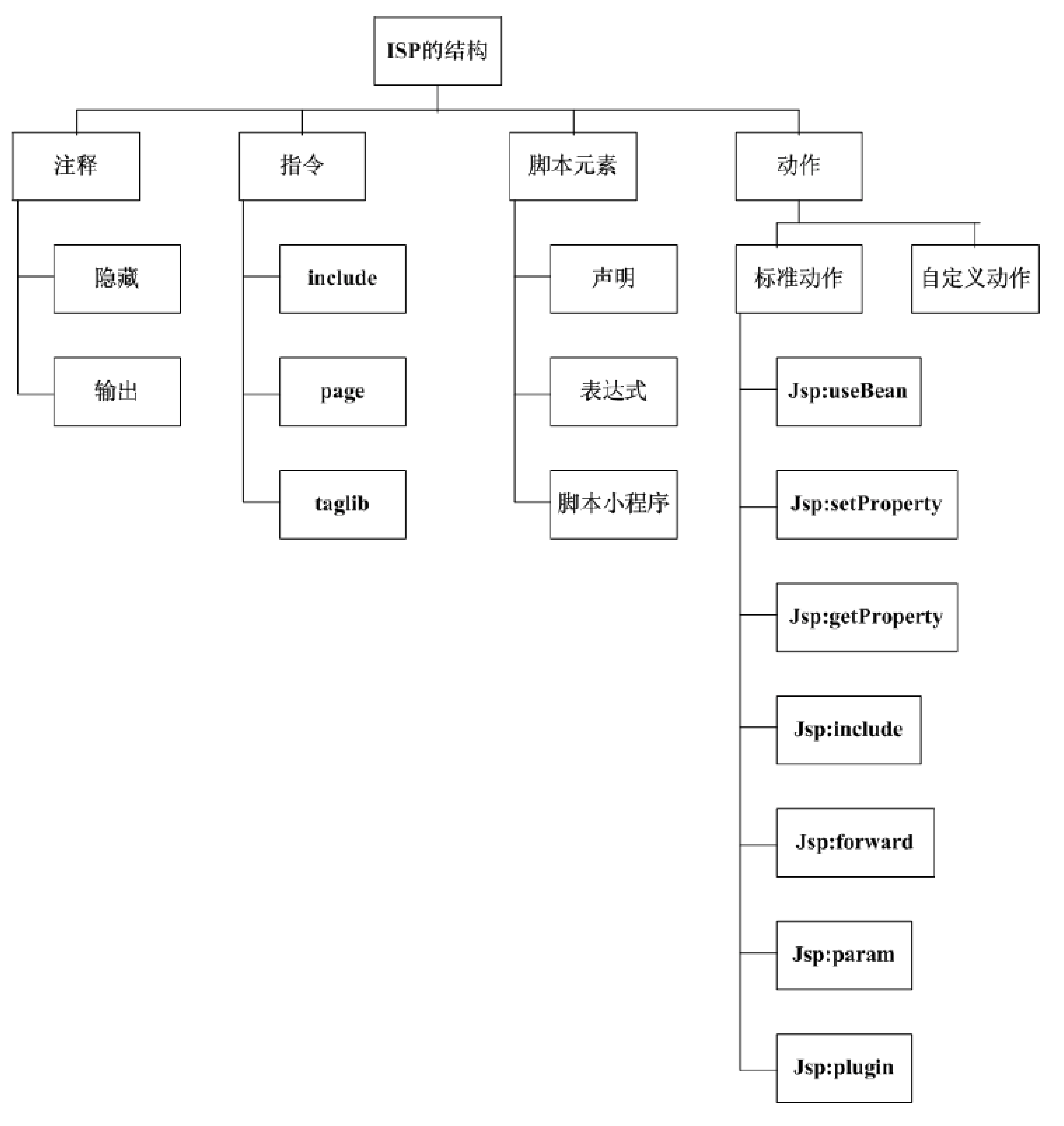


图 3-2　JSP 的组织结构

3.2　注释

JSP 页面中的注释有两种方式：一种是由页面生成的 HTML 注释；另一种是只在 JSP 页面中可视的隐藏注释。





3.2.1 HTML 注释

HTML 注释会在客户端的 HTML 源代码显示一个注释，其语法格式如下：

```
<!-- 注释[ <%= 表达式 %> ] -->
```

例子 1：

```
<!--这是一个 HTML 注释 -->
```

在客户端的 HTML 源代码中产生和上面一样的数据：

```
<!--这是一个 HTML 注释-->
```

> **提示**
>
> 这种注释和 HTML 代码的注释很像，也就是它可以在【查看源代码】中看到。惟一不同的是，可以在这个注释中用表达式(例子 2 所示)。这个表达式是不定的，由页面不同而不同，可以在此使用各种合法的表达式。

例子 2：

```
<!—页面生成于<%= (new java.util.Date()).toLocaleString() %>      -->
```

在客户端的 HTML 源代码中显示如下注释：

```
<!—页面生成于 2009 年 1 月 1 日 -->
```



3.2.2 隐藏注释

隐藏注释写在 JSP 程序中，但不发送到客户端，其语法格式如下：

```
<%-- 注释 --%>
```

例如：

```
<%@ page language="java" %>
<html>
  <head>
    <title>注释测试</title>
</head>
<body>
  <h2>这是一个注释的测试程序</h2>
  <%-- 这段注释在客户端应不可见--%>
  </body>
</html>
```



> **提示**
>
> 用隐藏注释标记的字符会在 JSP 编译时被忽略掉。这种注释在希望隐藏或注释 JSP 程序时是很有用的。JSP 编译器不会对<%--和--%>之间的语句进行编译，它不会显示在客户端的浏览器中，也不会在源代码中看到<%--和--%>之间的语句。



3.3 JSP 指令

JSP 指令是为 JSP 引擎而设计的，它们并不直接产生任何可见的输出，只是告诉引擎如何处理 JSP 页面。其通用的语法格式如下：

```
<%@ 指令名 [属性="值" 属性="值"…] %>
```

下面主要讨论 3 种标准 JSP 指令：page、include 和 taglib。

3.3.1 page 指令

page 指令设置影响到页面解释和执行方式的属性。在 page 指令中定义的属性适用于该 JSP 页面以及所有包含的静态文件，其基本语法格式如下：

```
<%@ page [属性="值" 属性="值"…] %>
```

表 3-1 列出了 Page 指令的属性。



表 3-1 page 指令属性

属 性	描 述
language	表示页面所使用的脚本语言，默认值为 java
extends	定义生成的服务器小程序(Servlet)的父类
import	Java 导入声明
session	决定 JSP 页面是否可以使用 SESSION 对象，默认值为 true
buffer	决定输出流是否有缓冲区，默认值为 8kb 的缓冲区
autoFlush	决定输出流的缓冲区是否要自动清除，默认值为 true
isThreadSafe	决定该 JSP 页面能否处理超过一个以上的请求，默认值为 true
info	表示该 JSP 网页的相关信息
errorPage	表示如果发生异常错误，网页将会重定向到由该属性指定的 URL
isErrorPage	表示该 JSP 页面是否为处理异常错误的网页
ContentType	表示 MIME 类型和 JSP 网页的编码方式

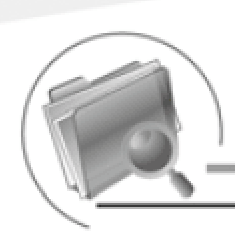
例如：

```
<%@ page import="java.util.*, java.lang.*" %>
<%@ page buffer="5kb" autoFlush="false" %>
<%@ page errorPage="error.jsp" %>
```

提示

<%@ page %>指令作用于整个 JSP 页面，同样包括静态的包含文件。但是<% @ page %>指令不能作用于动态的包含文件，如<jsp:include> 。

在一个页面中可以有多个<% @ page %>指令，但是其中的属性只能用一次，不过也有个例



外，那就是 import 属性。因为 import 属性和 Java 中的 import 语句差不多(参照 Java 语言)，所以在 JSP 页面中可以多次使用此属性。

无论<% @ page %>指令放在 JSP 的任何位置，它的作用范围都是整个 JSP 页面。不过，为了 JSP 程序的可读性，以及好的编程习惯，最好还是把它放在 JSP 文件的顶部。

3.3.2 include 指令

include 指令指示 JSP 容器在指令出现的位置包含一个特定的资源。包含的资源可以是 JSP 网页、HTML 网页、文本文件或者是一段 Java 程序。其语法格式如下：

```
<%@ include file="文件的路径" %>
```

include 指令只有一个 file 属性，该属性用于指定要包含的资源的路径。下面是一个简单例子：



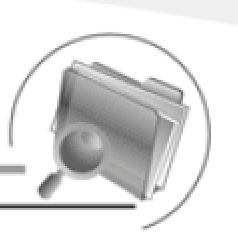

```
//ch3-1.jsp
<html>
  <head>
    <title>include 指令演示</title>
  </head>
  <body>
    下面将显示所要包含文件的内容：<br>
     <%@ include file="test.jsp" %>
  </body>
</html>
```

包含的 test.jsp 文件的代码如下所示：

```
<html>
  <head> </head>
  <body>
    当前的系统日期是：<%= new java.util.Date () %>
  </body>
</html>
```

有关 JSP 的表达式将在 3.4.2 节进行介绍。

运行 ch3-1.jsp，结果如图 3-3 所示。



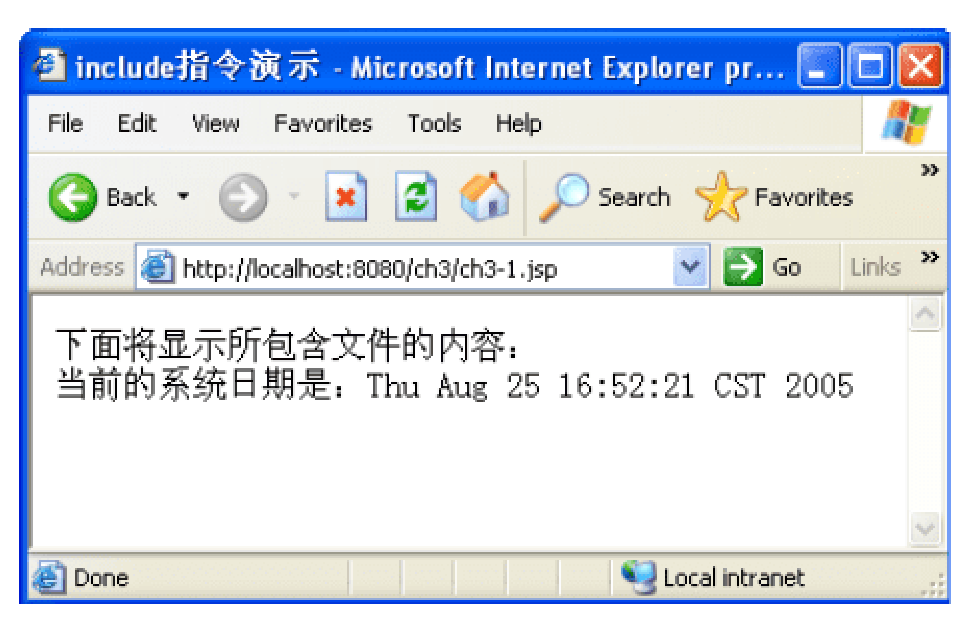


图 3-3　include 指令演示

3.3.3　taglib 指令

taglib 指令用于定义一个标签库以及其自定义标签的前缀，其语法格式如下：

```
<%@ taglib uri=" " prefix=" " %>
```

该指令包含如下两个属性：

- uri：指定标签库的路径。
- prefix：在自定义标签之前的前缀。

例如：

```
<%@ taglib uri="/tlbs/TableGenerator.tld" prefix="tg" %>
```

如果 TableGenerator.tld 标签库定义了一个名为 table 的标签，那么页面中就可以包含下述类型的标签：

```
<tg:table>
…
</tg:table>
```

有关 JSP 标签的内容将在本书后面章节中详细描述。

> **提示**
>
> <% @ taglib %>指令声明此 JSP 文件使用了自定义的标签，同时引用标签库，也指定了他们的标签前缀。
>
> 必须在使用自定义标签之前使用<% @ taglib %>指令，而且可以在一个页面中多次使用，但是前缀只能使用一次。

3.4　脚本元素

本节主要介绍 3 种脚本元素：声明、表达式和脚本小程序 scriptlet。可以使用声明来定义变量、声明方法、内部类或类一级的任何合法 Java 结构；可以向 JSP 表达式中插入任何合法的 Java 表达式；脚本小程序向 JSP 页面中插入任何有效的 Java 代码片断。



3.4.1 JSP 声明

在 JSP 程序中声明合法的变量和方法的语法格式如下：

```
<%! 声明 (s) %>
```

一个 JSP 页面中可以插入任意多个 JSP 声明，在 JSP 声明中插入任意多个 Java 声明。如下所示。

- 变量声明：

```
<%! int a = 0 %>
<%! Circle b = new Circle(2,0) %>
```

- 方法声明：

```
<%!
  public String fun(){
    return ("函数 fun() ! ");
  }
%>
```

- 内部类的声明：

```
<%!
  class InnerClass{
    public String Count( int num){
        return ("InnerClass [" + num + "]" );
    }
  }
%>
```

3.4.2 表达式

在 JSP 页面中包含一个符合 JSP 语法的表达式的语法格式如下：

```
<%= 表达式 %>
```

表达式可以很简单也可以很复杂，它可以是一个变量、算术表达式或方法调用等。

如下所示：



提示

不能用分号(";")来作为表达式的结束符。有时候表达式也能作为其他 JSP 元素的属性值。



```
<%= a %> //a 是已经声明并赋值的变量
<%= (1+2)*4 %>
<%= fun() >
```

3.4.3　脚本小程序 Scriptlet

Scriptlet 是用于处理 HTTP请求的一个或多个 Java 语句的集合，包含一个有效的程序段，其语法格式如下：

```
<% 程序段；[程序段；……] %>
```

JSP 编译器在_jspService ( )方法的主体中不修改包含 Scriptlet 的内容。JSP 页面可以包含任意数目的 Scriptlet。如果存在多个 Scriptlet，则每一个都附加到_jspService ( )方法中，并按其编号排序。因此，一个 Scriptlet 可以包含被括在大括号中的另外一个 Scriptlet。如下所示的是一个华氏温度到摄氏温度的转换表：

```
    <% page import = "java.text.*" %>
    <TABLE BORDER=0 CELLPADDING=3>
    <TR>
        <TH>Degrees<BR>Rahrenheit </TH>
        <TH>Degrees<BR>Celsius </TH>
    </TR>
<%
NumberFormat.fmt = new DecimalFormat ("### .000");
For (int f = 32;f<=212;f+=20){
    Double c = ((f-32)*5) /9.0;
    String cs = fmt.format(c);
  %>
        <TR>
            <TD ALIGN = "RIGHT"><% = f %></TD>
            <TD ALIGN = "RIGHT"><% = cs %></TD>
        </TR>
<%
}
%>
</TABLE>
```

实例代码包含两个 Scriptlet：一个对应循环主体；一个对应大括号。在两个 Scriptlet 之间是表格行的 HTML 标记，使用 JSP 表达式访问其值。生成的 Scriptlet 代码将 Scriptlet 内容转换，其代码如下：

计算机基础与实训教材系列

```
NumberFormat fmt = new DecimalFormat("###.000");
For (int f = 32 ;f<=222;f+=20){
    Double c = ((f-32)*5)/9.0;
    String cs = fmt.format(c);
    out.write("\r\n<TR>\r\n<TD ALIGN=\"RIGHT\">");
    out.print(f);
    out.write("</TD>\r\n");
    out.write("\r\n<TD ALIGH=\"RIGHT\">");
    out.print(cs);
    out.write("</TD>\r\n");
    out.write("</TR>\r\n");
}
```

输出如表 3-2 所示。

表 3-2 温 度 转 换

华 氏 度	摄 氏 度
32	.000
52	11.111
72	22.222
92	33.333
112	44.444
132	55.556
152	66.667
172	77.778
192	88.889
212	100.000

3.5 JSP 操作

JSP 操作将代码处理程序与特殊的 JSP 标记关联在一起。根据可扩展标记语言(eXtensible Markup Language, XML)规范，这些标记有两种可能的格式：

<前缀：标记名 [属性名=""…]> 标记主体 </前缀：标记名>
<前缀：标记名 [属性名=""…] />

第一种格式包括一个开始标记、零个或多个标记属性、一个标记主体以及一个包括前缀和标记名的结束标记；第二种格式包括一个开始标记、零个或多个标记属性以及一个结束标记。第二种格式正好相当于标记主体为空白的第一种格式。在解释一个 JSP 页面时，JSP 容器会在



遇到这些特殊标记时调用相关联的处理程序。

JSP 规范要求 JSP 容器支持一组标准的 JSP 操作以及一种开发自定义操作(标记库)的机制。本节主要介绍标准操作，所有的标准操作都使用保留的前缀 jsp，下面分别介绍这些标准操作，操作的语法格式一般按照第一种方式介绍。

⦿ <jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>操作

这 3 个操作主要用于 JSP 对 JavaBean 的支持，将在本书第 7 章中详细描述。

⦿ <jsp:include>操作

<jsp:include>操作允许包含动态和静态文件，这两种产生的结果是不尽相同的。如果包含的是静态文件，那么只是把它的内容加载到 JSP 网页中；如果包含的是动态文件，那么这个被包含的文件也会被 JSP 容器编译执行。

<jsp:include>操作的语法格式如下：

```
<jsp:include page="{relativeURL | <%= expression%>}" flush="true | false" >
标记主体
</jsp:include>
```



<jsp:include>有两个属性：page 和 flush。page 可以代表一个相对路径，即要包含的文件位置；flush 的值为布尔类型，若为 true 则表示缓冲区满时将会被清空，其默认值为 false。

如下所示：

```
<jsp:include page="scripts/Hello.jsp" />
<jsp:include page="scripts/login.jsp" >
  <jsp:param name="username"    value="administrator" />
  < jsp:param name="password" value="123456" />
</jsp:include>
```

<jsp:param>用于传递一个或多个参数给 JSP 网页，它的用法将在后面介绍。

⦿ <jsp:forward>操作

<jsp:forward>操作用于将客户端发出的请求从一个 JSP 网页转交给另一个 JSP 网页，其语法格式如下：

```
<jsp:forward page={"relativeURL" | "<%= expression %>"} >
标记主体
</jsp:forward>
```

<jsp:forward>操作只有一个 page 属性，它可以是一个相对路径，即要重新定向的网页位置，也可以是经过表达式运算的相对路径。

如下所示：

```
<%
  out.println("可以被执行！");
```

```
%>
<jsp:forward page="hello.jsp" >
   <jsp:param name="username" value="Administrator">
</jsp:forward>
<%
   out.println("不会被执行! ");
%>
```

在运行上面例子时，会打印出“可以被执行！”，之后转入 hello.jsp 页面，所以后面的“out.println("不会被执行! ")”语句将不被执行。使用<jsp:param>可以向目标文件传送参数和值。

◉ <jsp:param>操作

<jsp:param>操作用于向一个动态文件发送一个或多个参数，如果想传递多个参数，可以在一个 JSP 文件中使用多个<jsp:param>。其语法格式如下：

```
<jsp:param   name="parameterName" value={"parameterValue" | "<%= expression %>"} />
```

<jsp:param>操作有两个属性：name 和 value。name 指参数的名称；value 是参数的值。

◉ <jsp:plugin>、<jsp:params>和<jsp:fallback>操作

<jsp:plugin>元素用于在浏览器中播放或显示一个对象(典型的对象是 applet 和 bean)。当 JSP 文件被编译送往浏览器时，<jsp:plugin>将会根据浏览器的版本替换成<object>或者<embed>元素。

一般来说，<jsp:plugin>会指定对象是 applet 还是 bean，同样也会指定 class 的名字和位置，另外还会指定将从哪里下载这个 Java 插件。

> **提示**
>
> <object>用于 HTML 4.0，<embed>用于 HTML 3.2。

其语法格式如下：

```
<jsp:plugin type="bean | applet" code="classFileName"
            codebase="classFileDirectoryName"
   [ name="instanceName" ]
   [ archive="URIToArchive, ..." ]
   [ align="bottom | top | middle | left | right" ]
   [ height="displayPixels" ]
   [ width="displayPixels" ]
   [ hspace="leftRightPixels" ]
   [ vspace="topBottomPixels" ]
   [ jreversion="JREVersionNumber" ]
   [ nspluginurl="URLToPlugin" ]
   [ iepluginurl="URLToPlugin" ] >
  [ <jsp:params>
 [ <jsp:param name="paramName" value="{paramValue | <%= expression %>}" /> ]+
</jsp:params> ]
```



```
    [ <jsp:fallback> text message for user </jsp:fallback> ]
</jsp:plugin>
```

表 3-3 描述了<jsp:plugin>的属性。

表 3-3　<jsp:plugin>的属性

属　　性	描　　述
type	指定将被执行的插件对象的类型，必须指定是 bean 还是 applet，因为该属性没有默认值
code	将会被 Java 插件执行的 Java Class 的名字，必须以.class 结尾，且该文件必须存在于 codebase 属性指定的目录中
codebase	将会被执行的 Java Class 文件的目录(或者是路径)，如果没有提供此属性，那么将使用<jsp:plugin>的 JSP 文件的目录
name	指定这个 Bean 或 applet 实例的名字，它将在 JSP 页面的其他地方调用
archive	一些由逗号分开的路径名，这些路径名用于预装一些将要使用的 class，这会提高 applet 的性能
align	图形、对象、applet 的位置
height、width	applet 或 bean 将要显示的长宽值，此值为数字，单位为像素
hspace、vspace	applet 或 bean 显示时在屏幕左右、上下所需留下的空间，单位为像素
Jreversion	applet 或 bean 运行所需的 Java Runtime Environment (JRE) 的版本
nspluginurl	Netscape Navigator 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL，如 http://www.aspcn.com/jsp
iepluginurl	IE 用户能够使用的 JRE 的下载地址，此值为一个标准的 URL，如 http://www.aspcn.com/jsp



标记主体描述<jsp：params>的语法格式如下：

```
<jsp:params>
  [<jsp:param name="paramName" value="{paraValue|<%=expression%>}" /> ]+
</jsp:params>
```

它用于指定向 applet 或 bean 传送的参数或参数值。

<jap：fallback>元素用于当客户端浏览器不能启动 applet 或 bean 时，浏览器要显示的错误信息，其语法格式如下：

```
<jsp:fallback> 给用户的文本信息 </jsp:fallback>
```

例如：

```
<jsp:plugin type="applet" code="Test.class" codebase="/scripts">
  <jsp:params>
     <jsp:param name="username" value="Administrator" />
```




```
    </jsp:params>
    <jsp:fallback>
      <p> 不能启动 applet！ </p>
    </jsp:fallback>
  </jsp:plugin>
```

3.6 实例

下面的实例融入了本章介绍的大部分语法，其功能是向客户端浏览器传回一个 HTML 表格，其中包括浏览器发送的 HTTP 请求头标：

```
//ch3-2.jsp
  <%@ page language="java" contentType="text/html;charset=gb2312" %>
  <%@ page import="java.util.*" %>
<html>
  <head>
    <title>JSP 语法示例</title>
    <style>
      <jsp:include page="style.css" flush="true" />
    </style>
  </head>
  <body>
    <h2>接收到的 HTTP 请求头信息</h2>
    <table border="1" cellpadding="4" cellspacing="0" >
    <%
      Enumeration Names = request.getHeaderNames();
      while( Names.hasMoreElements() )
      {
        String    name = (String)Names.nextElement();
        String    value = request.getHeader(name);
      // String value = "tet";
    %>
      <tr>
        <td> <%= name %> </td>
        <td> <%= value %> </td>
      </tr>
    <%
      }
    %>
  </body>
</html>
```



运行结果如图 3-4 所示。

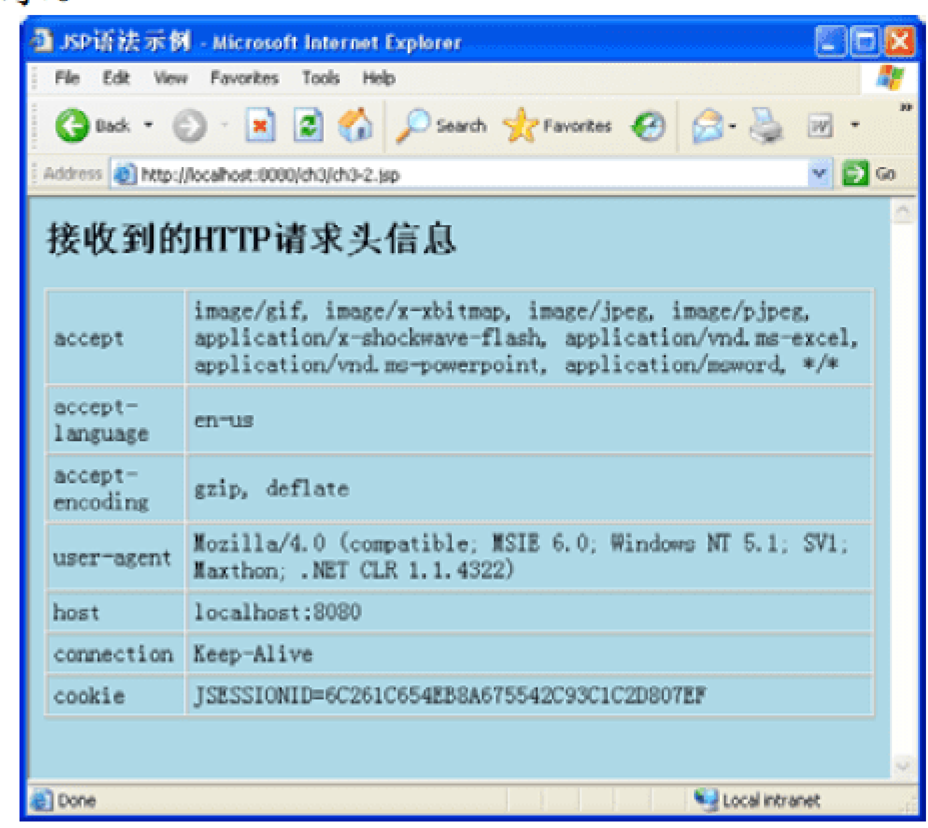
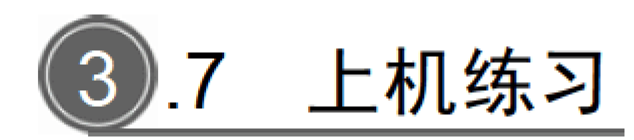


图 3-4 JSP 语法示例



3.7 上机练习



本章上机练习主要练习如何使用 Eclipse 创建 Tomcat 项目以进行 JSP 页面编程。其中重点掌握如何在 Eclipse 中创建 Tomcat 项目，学习并熟悉 JSP 的各种语法等。

下面以 3.6 节的 JSP 页面为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在的文件夹，双击 Eclipse.exe 文件，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮，Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中会出现新的 testTomcat 项目。

(6) 选择【File】|【New】|【Other】命令，弹出 New 对话框新建一个 JSP 文件。

(7) 在 New 对话框中选择【Web】|【JSP】选项，单击 Next 按钮弹出 New JavaServer Page 对话框。

(8) 在 New JavaServer Page 对话框中的 File name 文本框中输入文件名，如 ch3-2.jsp。

(9) 单击 Finish 按钮即可新建 JSP 页面文件。

(10) Eclipse 会创建一个新的文件，包括了基本的页面框架。在编辑窗口中输入 ch3-2.jsp 页面的代码。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移到错误位置，会显示相应的错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(14) 打开 Internet Explorer 浏览器，输入相应的 URL，如：http://localhost:8080/testTomcat/ch3/ch3-2.jsp。观察页面是否正常运行。

3.8 习题

3.8.1 填空题

1. 在 JSP 中，对象的作用域有________________、_________________、_________________和_________________。

2. JSP 的注释类型有_________________和_________________。

3. JSP 指令有____________________、_______________和_____________。

4. JSP 的脚本元素有_________________、_________________和_____________。

3.8.2 选择题

1. 在 page 指令中，(　　)属性是可以在页面中重复的属性。

A. language　　B. buffer　　C. import　　D. autoFlush

2. JSP 的结构由(　　　)组成(多选)。

A. 指令　　B. 注释　　C. 脚本元素　　D. 操作

3. 以下选项中，哪一项不是 JSP 指令(　　)。

A. page　　B. import　　C. include　　D. taglib

3.8.3 问答题

1. 首次加载 JSP 页面时，将经历哪几个阶段？

2. taglib 指令的作用是什么？



Java 编程语言

学习目标

Java 是由美国 Sun 公司开发的支持面向对象程序设计的语言，它借助于虚拟机机制实现跨平台特性使得 Java 迅速流行。目前，随着 J2ME、J2SE 和 J2EE 的发展，Java 已经不仅仅是一门简单的计算机开发语言了，它已经拓展出一系列的业界先进技术。

JSP 作为 J2EE 的一项关键技术，是 Java 技术的一个重要组成部分。在 JSP 中的逻辑实现部分都是采用符合 Java 语法的语句，同时 JSP 用到的 JavaBean 以及 Servlet 技术都是标准的 Java 程序，所以在深入学习 JSP 之前，首先必须掌握 Java 语言的基本语法以及 Java 面向对象编程的基础知识。

本章重点

- Java 的数据类型
- 流程控制和跳转语句
- 在 Eclipse 中开发 Java 程序

4.1　Java 概述

在 1995 年 5 月 23 日，JDK(Java Development Kits)1.0a2 版本正式对外发表，它标志着 Java 的正式诞生。随后，Java 凭借其所具备的优点迅速流行起来。简单说来，Java 语言具有如下所述的一些特点。

- 平台独立性

平台独立性意味着 Java 可以在支持 Java 的任何系统上“独立于所有其他软硬件”而运行，例如，不管系统装的是 Windows、Linux、Unix 还是 Macintosh，也不管机器是大型机，小型机还是微机，甚至是 PDA 或者手机、智能家电，Java 都能运行，当然这里有一点需要指出的是：

在这些平台上都应装有相应版本的 JVM(Java 虚拟机)，即平台必须支持 Java！

现在大家的手机很多都是支持 Java 的，而且大多数手机游戏也都是 Java 开发的，这样任何支持 Java 的手机都能玩这些游戏，这也是平台独立所带来的好处吧。平台独立保证了软件的可移植性，而软件的可移植性是软件投资在未来的保证。用 Java 写软件保证了程序在将来无需再移植。可移植性是业界一直以来所宣扬的最大卖点和亮点，但以前从未实现过，是 Java 使软件行业真正实现了软件的可移植性。

⦿ 安全性

现今的 Java 语言主要用于网络应用程序的开发，因此对安全性有很高的要求。如果没有安全保证，用户运行从网络下载的 Java 语言应用程序是十分危险的。Java 语言通过一系列的安全措施，在很大程度上避免病毒程序的产生和网络程序对本地系统的破坏，具体有：

(1) 去除指针这种数据类型，简化编程，更是避免了对内存的非法访问。

(2) Java 是一种强类型的程序设计语言，要求显示的声明，保证编译器可以提前发现程序错误，提高程序可靠性。

(3) 垃圾自动回收机制让程序员从烦人的内存管理工作中解脱出来，专注于程序开发，更重要的是，通过这种内存自动回收机制，可以很好的确保内存管理的正确性，避免出现“内存泄露”现象。

(4) Java 语言提供了异常处理机制。

(5) Java 程序运行时，解释器会对其进行数组和字符串等的越界检查，确保程序安全。

⦿ 多线程

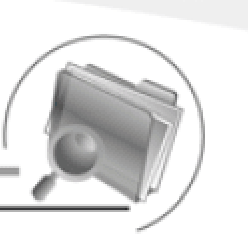
在 DOS 年代里，人们一次只能运行一个程序，执行完才能运行下一个，后来出现了视窗 windows 之后，人们可以同时运行几个程序，并在各个运行程序间做切换，比如一边听音乐一边编辑 word 文档，这时的操作系统出现了进程的概念，每个运行中的程序都是一个进程，再后来，为了提高程序的并发性，又引入了线程的概念，线程也称作轻量级进程，进程是系统分配资源的基本单位，而线程成为了系统 CPU 调度执行的基本单位，一个进程可以只有一个线程，也可以有多个线程，在很多情况下，开发多线程的程序还是很有必要的，幸运的是 Java 对多线程编程提供了很好的支持。

⦿ 网络化

在网络环境中，对象可以在本地或远程机器执行。Java 程序可以通过网络打开和访问对象，就像访问本地系统一样简单。Java 语言提供的丰富类库保证了其可以在 HTTP、FTP 和 TCP/IP 协议中良好运行。Java Applet 程序需要客户端浏览器的支持，并且其是通过标签对<applet>< /applet >将自己嵌入 HTML 中。当用户浏览该 Web 页时，Java Applet 程序才从服务器端下载到客户端解释执行。因此也称 Java Applet 是可移动代码，这种移动性为分布式程序开发提供了一种新的技术途径。

⦿ 面向对象

随着软件业的发展，面向对象的程序设计方法已经流行起来，出现了很多面向对象的编程语言，如 Java、C++等。这里需要指出的是面向对象的程序设计方法与编程语言之间是不同的概念，

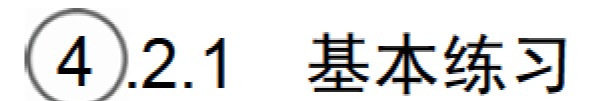


面向对象的程序设计不一定非得用面向对象的语言才能实现，反过来，用面向对象的编程语言写出来的程序也未必就是面向对象的。以面向对象的思想进行程序设计，并用面向对象的编程语言进行开发实现是当下软件开发领域最常被采用的。简单说，面向对象主要是由于引入了类这个重量级的“数据类型”，使得原本的面向过程程序设计有了质的飞跃，其实类这个新类型中不仅仅包含数据部分，它还包含操作方法(也可以管它叫函数)，这个囊括了数据和算法的类成为面向对象程序设计中最关键的要素，可以说，所有设计开发任务都是围绕类而展开的，同样，面向对象技术的特征也是由类体现出来的，最主要的 3 大特征是：封闭性、继承性和多态性。

4.2 Java 数据类型

正如人们在认识客观事物时总是将其分门别类一样，计算机在处理各种各样的数据时也需要区分它们的类型。在 Java 中，数据类型被划分为基本类型和引用类型两大类。基本类型是 Java 内置的预定义类型，包括数值类型(整数或浮点数)、布尔类型和字符类型。引用类型则包括类、接口和数组等。





4.2.1 基本练习

数据类型代表了数据的存储格式和处理方式，虽然严格来说计算机只能识别 0 和 1，但是，有了数据类型以后，计算机的识别能力就被人为扩展了，它能够识别整数、实数以及字符等，比如整数 55，实数 75.5，字符 a 或 A 等，Java 提供了 8 种基本数据类型，它们在内存中所占据的存储空间如表 4-1 所示。这 8 种基本数据类型可以分为以下 4 组：

布尔型：boolean

整型：byte、short、int 以及 long

浮点型(实型)：float 以及 double

字符型：char

表 4-1 Java 的基本数据类型

类型名称	类型标识	占据存储空间	取值范围
布尔型	boolean	1bit	true 或 false
整型	byte	8bits(1Byte)	-128～+127
	short	16bits(2Bytes)	-32768～+32767
	int	32bits(4Bytes)	-21 亿 ～+21 亿
	long	64bits(8Bytes)	-9.2×10^{18}～$+9.2\times10^{18}$
浮点型	float	32bits(4Bytes)	7 位精度
	double	64bits(8Bytes)	15 位精度
字符型	char	16bits(2Bytes)	Unicode 字符

下面对该 8 种基本数据类型分别进行介绍。首先来看一下最简单的布尔型。

⊙ 布尔型

布尔类型用关键字 boolean 来标识，其取值范围就是：true(逻辑真)和 false(逻辑假)，是最简单的数据类型。布尔类型的数据可以参加逻辑运算，并构成逻辑表达式，其结果也是布尔值，常用来作为分支、循环结构中的条件表达式。

```
boolean flag1 = true;
boolean flag2 = 3>5;
boolean flag3 = 1;
```

上面定义了 3 个布尔类型的变量 flag1、flag2 和 flag3，其中 flag1 直接初始化为 true 值，而 flag2 的初值为 false(因为关系运算 3>5 的结果为假)，flag3 的值 true(因为 Java 语言规定可以用 0 代表假，非 0 代表真)。

⊙ 整型



用关键字 byte、short、int 和 long 标明的数据类型，都是整数类型，简称整型。整型的值可以是正整数、负整数或者整数零，比如 222，-211，0，2000，-2000 等都是合法的整型值，而 222.2，2a2 等是非法的，222.2 有小数点，不是整型，2a2 含有非数字字符，亦不可能是整型值。在 Java 语言(包括大多数编程语言)中，常量值一般默认以十进制进行表示。另外，有一个注意的问题是：由于数据类型的存储空间大小是有限的，因而其所能表达的数值大小也是有限的，即每一种数据类型都对应有一个取值范围，一般存储空间大的，其值域也大，如整型的 4 种具体类型中，byte 的取值范围最小，而 long 类型的最大。下面分别对每一种整型数据类型进行介绍。

(1) byte

byte 类型是整型中最小的，它只占据 1 字节的存储空间，由于采用补码方式，其取值范围为-128~127，适合于用来存储诸如此类的数据：人的年龄、定期存款的存储年限、图书馆借书册数、楼层数等等，这类数据一般取值都在该范围之内。若用 byte 变量来存放较大的数，如：

```
byte rs = 10000；//定义 rs 变量存放清华大学的学生人数
```

就会产生溢出错误，即 byte 的变量无法存放(表达)10000 这么大的数，解决办法是用更大的空间来存放，也就是说将 rs 变量定义为较大的数据类型，如以下 short 类型。

(2) short

short 整型可以存放的数值范围为：-32768～+32767，因而其可以胜任。

```
short rs = 10000；//正确
```

一个 short 类型的整型变量(如上 rs)占据的空间为 2 个字节，占据的空间大了，其表示能力(取值范围)自然就大。同样的，假如 rs 变量要用来存放当前全国高校的在读大学生数量，则 short 类型又不够了，需用更大的，如 int 类型。



(3) int

int 占据 4 个字节空间，可以存储大概在-21 亿~21 亿范围间的任意整数。该类型在程序设计中是较常用的类型之一，且程序中整型常量的默认数据类型就是 int，因为一般情况，int 就够用了，但是现实生活中，还是有不少情况需要用到更大的数，比如世界人口，某银行的存款额，世界巨富的个人资产，某股票市值等等，所以 Java 还提供了更大的整型 long。

(4) long

long 占据 8 个字节，能表示的数值范围多达$-9.2\times10^{18}\sim+9.2\times10^{18}$。一般如不是应用需要，尽量少用，可以减少存储空间的支出。当然，long 也不是无限的，在一些特殊领域，如航空航天，它也可能会不够用，这时可以通过定义多个整型变量来组合表示这样的数据，即对数据进行分段表示，不过，在真正的实践中，这些领域的计算任务一般会由支持更大数据类型的计算机系统来完成，譬如大型机，巨型机。

需要指出的是，整型变量的类型并不直接影响其存储方式，类型只决定变量的数学特性和合法的取值范围。如果对变量赋了超出其取值范围的值，Java 编译系统会进行错误提示，尽管如此，大家在进行程序设计时，还是应主动加以避免。



⊙ 浮点型

浮点型有两种，分别用关键字 float 和 double 来标识，其中 double 的精度较高，表数范围也更广。float 被称为单精度浮点型，double 为双精度浮点型，程序中出现的浮点数在默认情况下即为 double 类型。

⊙ 字符型

Java 语言用 Unicode 字符集来定义字符型，因此一个字符需要两个字节的存储空间，这点与 C/C++不同，需要注意。下面来看字符型的变量定义。

```
char ch;   //定义字符型变量 ch
ch = '1';   //给 ch 赋初值为'1'
```

字符型变量在程序常被用作代号，比如 ch 为 1 代表成功，为 0 代表失败；为 F 表示女性，为 M 表示男性等。在具体进行程序设计时，应注意灵活应用。

4.2.2　引用类型

引用类型主要包括类类型和数组等。

⊙ 类(class)类型

class 类型定义了一种数据结构，该数据结构中包含了数据成员和函数成员。class 在 OOP(Object Oriented Programming，面向对象编程)中是非常重要的概念，关于 class 的详细内容将在本书第 5 章中详细讨论。在此之前可以简单地将 class 理解为一种可以自定义的复杂类型。

⊙ 数组

数组是包含相同类型变量的集合。数组元素的类型可以是任何类型，包括基本类型、类类型甚至数组类型。数组用下标来确定每一个元素的索引号，通过索引来访问其成员变量的值。只有一个下标的数组称为一维数组，多于一个下标的则称为多维数组。

```
int a[];      //int 型的一维数组
int b[][];    //int 型数组的数组
```

为了更清晰地表示变量的类型是数组，Java 也允许采用将变量名放在方括号后面的方式进行声明，如：

```
int[] a;
```

上面的语句只是声明了一个数组，指定这个数组中元素的类型。如果要使用这个数组，还需要指定它的元素个数，Java 中使用 new 运算符来创建数组：



```
int[] a = new int[10];  //创建一个包含 10 个元素的整数数组
```

当使用上述语句创建一个新的数组时，数组中所有元素都被初始化。如上述整数数组 a 的每个元素的值都会被初始化为 0。操作数组时需要注意的是，用于指示单个数组元素的下标必须总是从 0 开始，并保持在合法范围之内(大于或等于 0 并小于数组长度)。任何访问在上述界限之外的数组元素的企图都会引起运行时错误。例如下面的语句可以对 a 数组的数据进行更新：

```
a[0]=1;
a[1]=2;
…
a[9]=10;
a[10]=11;      //错误，对数组 a 的访问越界
```

另外一种比较简便的创建并初始化数组的方法是在声明数组的同时直接进行赋值，如：

```
int[] array1 = {1, 2, 3, 4};
```

上述语句会创建一个包括 4 个元素的整数数组 array1，并对每一个元素进行初始化，其中 array1[0]等于 1，array1[1]等于 2，等等。



提示

数组类型是一种引用类型，所以一个数组实际上是一个对象(本书第 5 章详细介绍)。Java 编译器不会在声明时自动初始化数组，因此需要确保在使用数组之前对其进行初始化。另外，由于数组类型是一个对象，因此也赋予了其一些属性，这些属性对于访问或操作一个数组很有帮助。可以使用 length 属性来确定一个数组的大小，如 array1.length 等于 4。



4.3　符号

每一个 Java 程序都是按照一定规则编写而成，这些规则一般称之为程序语法，只有语法正确了，程序才能通过编译系统的编译，进而也才能被计算机加以执行，本节向读者介绍 Java 程序的符号概念。

4.3.1　基本符号元素

字母：A-Z，a-z，美元符号$和下划线(__)。

数字：0-9。

算术运算符：+，-，*，/，%。

关系运算符：>，>=，<=，!=，==。

逻辑运算符：!，&&和||。

位运算符：~，&，|，^，<<，>>，>>>。

赋值运算符：=。

其他符号：()，[]，{}等等。



4.3.2　关键字

关键字是 Java 语言本身使用的标识符，有其特定的作用。所有的 Java 关键字将不能被用作用户的标识符，关键字用英文小写字母表示。

Java 关键字有：

abstract	else	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws
char	for	private	transient
class	if	protected	true
countinue	implements	public	try
default	import	return	void
do	instanceof	short	while
double	int	static	

4.3.3 标识符

标识符特指用户自定义的标识符。在 Java 语言中，标识符必须以字母、美元符号或者下划线打头，后接字母、数字、下划线或美元符号串。另外，Java 语言对标识符的有效字符个数不做限定。

合法的标识符：

a，b，c，x，y，z，result，sum，value，a2，x3，_a，$b 等。

非法的标识符：

2a，3x，byte，class，&a，x-value，new，true，@www 等。

为了提高程序可读性，以下特别列举几个较为流行的标识符命名约定：

(1) 一般标识符定义应尽可能"达意"，如 value，result，number，getColor，getNum，setColor，setNum 等。

(2) final 变量的标识符一般全大写，如 final double PI=3.1415。

(3) 类名一般用大写字母打头，如 Test，Demo。

4.3.4 分隔符

Java 语言中，分隔符大致可以分为以下两大类：

⊙ 空白符

空白符在程序中主要起间隔作用，没有其他的意义，因此编译系统利用其区分完程序元素后，即将其忽略掉。空白符包括空格、制表符、回车和换行符等，程序各基本元素间通常用一个或多个空白符进行间隔。

⊙ 可见分隔符

可见分隔符也是用来间隔程序基本元素的，这一点同空白符类似，但是不同的可见分隔符有不同的用法。Java 语言中，主要有 6 种可见分隔符：

(1) //

用来注释程序用的，从该符号后的本行内容均为注释，辅助程序员阅读程序，而被编译系统忽略，没有其他意义。也称单行注释符。

(2) /*和*/

/*和*/是配对使用的多行注释符，以/*开始，至*/结束的部分均为注释内容。

(3) ；

分号用来标识一个程序语句的结束，因此在每一个语句编写完后，记得添加语句结束标志——分号，这点是多数初学者容易遗忘的。



(4) ，

逗号一般用来间隔同一类型多个变量的声明，或者间隔方法中的多个参数。

(5) ：

冒号可以用来说明语句标号，或者用于 switch 语句中的 case 分句。

(6) {和}

花括号也是成对出现的，{标识开始，}标识结束，可以用来定义类体、方法体、复合语句或者进行数组的初始化等。

4.4　程序语句

一个 Java 程序其实是由若干条语句组成的，而语句又是由表达式或简单的声明组成，本节将介绍这些基本的概念。



4.4.1　赋值语句

赋值语句的一般形式为：

```
variable = expression;
```

在这里=不是数学中的等号，而是赋值运算符，这点初学者务必牢记，其功能是将右边表达式的值赋(即传递或存入)给左边的变量，例如：

```
int  i，j;
char c;
i = 100;
c = 'a'
j = i +100;
i = j * 10;
```

第一个赋值语句将整数 100 存入 i 变量的存储空间，第二个将字符常量'a'存入字符变量 c，第三个则首先计算表达式 i+100 的值，i 变量此时存放的值为 100，因此该表达式的值显然为 100+100，即 200，然后再将表达式值 200 存放至另一变量 j 的空间中，第四个赋值语句同样先计算右边表达式的值，计算后值为 2000，然后再将其存放至 i 变量的空间中，注意此时 i 变量的值变为 2000 了，原本的值 100 也就不复存在了，或者说是旧值被新值覆盖了。

特别地，对于形如 i=i+1；这样的赋值语句，可以将其简写为 i++；或者++i，并称之为自增语句，同样还有自减语句 i--；或者--i；，它们等价于 i=i-1；语句。把++和--叫做自增和自减，它们写在变量的前面与后面有时是有区别的，请看下例。

【例 4-1】自增赋值语句。

```
public class Test
{
    public static void main(String[] args)
    {
        int i, j , k = 1;
        i = k++;
        j = ++k;
        System.out.println("i="+i);
        System.out.println("j="+j);
    }
}
```

程序运行结果如下：

```
i = 1
j = 2
```

当自增符号++写在后面时，先访问后自增，即 i = k++;语句其实等价于 i=k；和 k++；两条语句；而自增符号++写在前面时，则先自增后访问，即 j = ++k;语句相当于++k；和 j=k；两条语句，因此得到上述程序的运行结果。这点对于自减语句也是一样的。

下面再介绍一下复合赋值语句，常用的复合赋值运算有：

+=　　加后赋值
-=　　减后赋值
*=　　乘后赋值
/=　　除后赋值
%=　　取模后赋值

【例 4-2】复合赋值语句。

```
public class Test
{
    public static void main(String[] args)
    {
        int i=0, j=30 , k = 10;
        i += k;          //相当于 i = i+k;
        j -= k;          //相当于 j=j-k;
        i *= k;          //相当于 i=i*k;
        j /= k;          //相当于 j=j/k;
        k %=i+j;         //相当于 k=k%(i+j);
        System.out.println("i="+i);
```



```
            System.out.println("j="+j);
            System.out.println("k="+k);
        }
    }
```

程序运行结果如下：

```
i=100
j=2
k=10
```

上述程序中 k %=i+j;语句等价于 k=k%(i+j)；语句，初学者常犯的错误是，将其等价于没有小括号的 k=k%i+j；语句，显然二者结果是截然不同。事实上，复合赋值语句仅是程序的一种简写方式，因此，建议初学者待熟练掌握编程后再采用此方式。

4.2　条件表达式



条件表达式的一般形式为：

Exp1？Exp2：Exp3

首先计算 Exp1，当表达式 Exp1 的值为 true 时，计算表达式 Exp2 并将结果作为整个表达式的值，当表达式 Exp1 的值为 false 时，计算表达式 Exp3 并将结果作为整个表达式的值，请看下例。

【例 4-3】条件表达式示例。

```
public class Test
{
    public static void main(String[] args)
    {
        int i, j=30 , k = 10;
        i = j==k*3?1:0;
        System.out.println("i="+i);
    }
}
```

程序运行结果如下：

```
i=1
```

表达式 Exp1：j= =k*3，其值为 true，因此，整个条件表达式的取值为 Exp2 之值，即 1。

4.4.3 运算

(1) 算术运算

Java 的算术运算有加(+)、减(-)、乘(*)、除(/)和取模(%)运算。前 3 种运算比较简单，但后两种则需要注意一下。当除运算符两边的操作数均为整数时，其结果也为整数，否则为浮点数，比如：

3/2　　结果为 1

3/2.0　　结果为 1.5

尤其当参与运算的操作数为变量时，更需要注意其数据类型对于结果的影响。此外，%为取模运算，即求余数运算，比如：

5%2　　结果为 1

11%3　　结果为 2

特别地，取模运算要求参与运算的操作数必须均为整数类型。

(2) 关系运算

关系运算的结果为布尔值，即 true 或 false，Java 语言中共有 6 种关系运算：>(大于)、>=(大于等于)、<(小于)、<=(小于等于)、= =(等于)、 ! =(不等于)。下面举一例子。

【例 4-4】关系运算示例。

```
public class Test
{
    public static void main(String[] args)
    {
        int i=0, j=30 , k = 10;
        boolean b1,b2,b3;
        b1 = i>k;
        b2 = i<=j;
        b3 = j/3!=k;
        System.out.println("b1="+b1+",b2="+b2+",b3="+b3);
    }
}
```

程序运行结果如下：

```
b1=false,b2=true,b3=false
```

(3) 逻辑运算

Java 语言中有 3 种逻辑运算：&&(与)、||(或)、!(非)，参与逻辑运算的操作数为布尔类型值，最终结果也为布尔值，其真值表如表 4-2 所示。

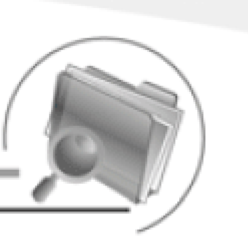


表 4-2　逻辑运算真值表

x	y	x&&y	x||y	!x
true	false	false	true	false
true	true	true	true	false
false	true	false	true	true
false	false	false	false	true

对于逻辑与运算，只要左边表达式值为 false，则整个逻辑表达式的值即为 false，不必再对右边表达式进行计算，同样，对于逻辑或运算，只要左边表达式值为 true，则整个逻辑表达式的值即为 true，不必再计算右边的表达式。

(4) 位运算

位运算指的是对二进制位进行计算，其操作数必须为整数类型或者字符类型。Java 提供的位运算如表 4-3 所示。

表 4-3　位运算

运　算　符	用　　法	功　　能
&	ope1 &ope2	按位与
|	ope1 |ope2	按位或
~	~ ope1	按位取反
^	ope1 ^ope2	按位异或
<<	ope1<<ope2	左移
>>	ope1>>ope2	带符号右移
>>>	ope1>>>ope2	不带符号右移

按位与、按位或以及按位取反运算都相对简单；按位异或运算规则为：0^0=0，0^1=1，1^0=1，1^1=0；左移运算将一个二进制数的各位全部左移若干位，高位溢出丢弃，低位补上 0；带符号右移运算中，低位溢出丢弃，高位补上操作数的符号位，即正数补 0，负数补 1；不带符号右移运算，低位丢弃，高位一概补 0。另外，需要特别提醒的是，当今绝大多数计算机的操作数都是以补码形式表示的，因此在进行位运算时要注意这一点。

(5) 运算优先级

其实，赋值和条件表达式也是一种运算的形式，各运算按照优先级递增排序为：赋值运算，条件运算，逻辑运算，按位运算，关系运算，移位运算以及算术运算。

4.4.4　复合语句

语句(statement)是程序的基本组成单元，在 Java 语言中，有简单语句和复合语句之分，一

条简单语句总是以分号结束，它代表一个要执行的操作，可以是赋值、判断或者跳转等语句，甚至还可以是只有分号的空语句(;)，空语句表示不需要执行任何的操作，而复合语句则是指用大括号括起来的语句块(block)，它一般由多条语句构成，但也允许只有一条简单语句。格式如下：

```
{
 简单语句 1;
 简单语句 2;
 …
 简单语句 n;
}
```

比如以下例子均为复合语句：

```
{
 a = 1;
 b = 2;
}
```

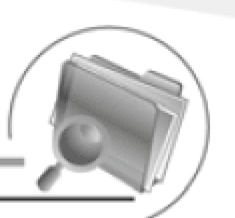
或

```
{
     S = 0;
}
```

复合语句在下面的流程控制结构中经常要用到，比如需要多个语句作为一个“整体语句”出现时就必须用大括号将其括起来作为一条复合语句。一般地，Java 程序的语句流程可以分为以下 3 种基本结构：顺序结构、分支(选择)结构以及循环结构。对于分支结构和循环结构，当条件语句或者循环体语句多于一条时，必须采用复合语句的形式，即用大括号将其括起来，否则系统将默认条件语句或循环体语句仅有一条，即最近的那一条。反过来说，当条件语句或者循环体语句只有一条时，则可用可不用大括号｛｝，这点请初学者学习后面内容时注意留心。

4.5 流程控制

由赋值语句以及输入输出语句构成的程序，只能按其书写顺序自上而下，从左到右依次执行，因此将此类程序结构称为顺序结构，这是最简单的程序结构，也是计算机执行的最通常流程。此外，流程控制语句还包括分支结构(if 语句和 switch 语句)、循环结构(while 语句，do-while 语句和 for 语句)和跳转语句(break 语句和 continue 语句)等，本节将对它们作简要介绍。



4.5.1　分支结构

分支结构表示程序中存在分支语句，这些语句根据条件的不同，将被有选择地加以执行，即可能执行，也可能不执行，这完全取决于条件表达式的取值情况。根据分支的多少，可以将其划分为：单分支结构、双分支结构以及多分支结构。Java 语言的单分支语句是 if 语句，双分支语句是 if-else 语句，多分支语句是 switch 语句，当然，实现时，也可以用 switch 语句构成双分支结构，或者用 if-else 语句嵌套构成多分支结构。下面分别对不同的分支结构进行介绍。

◉　单分支条件语句

单分支条件语句的一般格式是：

```
if (布尔表达式)
{
    语句;
}
```

当其中的语句仅为一条时，大括号可以缺省，但若为多条语句，则千万不能漏了大括号，否则，程序的涵义就变了，下面请看一个小程序段：

```
int i=0,j=0;
if (i!=j)
{   i++;
    j++;
}
```

以上程序段执行后 i，j 的值显然仍为 0，但是假如将 if 条件语句的大括号省掉，如下：

```
int i=0,j=0;
if (i!=j)
  i++;
  j++;
```

则执行后 i 的值仍为 0，而 j 的值则变为 1，即 j++; 语句被执行了，因为没有了大括号，if 的条件语句只由 i++; 单条语句构成，即使条件表达式取值为 false，j++; 语句也会被执行。另外，这种情况下的程序编排最好采用如下格式的缩进：

```
int i=0,j=0;
if (i!=j)
  i++;
j++;
```

以体现程序的层次结构，提高可读性。



◉ 双分支条件语句

Java 语言的双分支结构由 if-else 语句实现，一般格式如下：

```
if (布尔表达式)
   {
     语句 1；
   }
else
   {
     语句 2；
   }
```

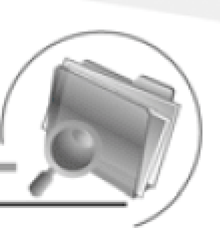
双分支语句在单分支结构基础上，增加了 else 结构，当布尔表达式为真时，执行语句 1 部分，否则为假时，执行语句 2 部分，不管布尔表达式取值如何，两部分语句必然有一部分被执行，语句 1 和语句 2 可以是多条语句，也可以是单条语句。多条语句时，别忘了用{}将其括起来整体作为一条复合语句；而单条语句时，建议初学者最好也写上{}，等用熟练了，再将其缺省掉。请看以下程序段：

```
int i=0,j=0;
if (i==j)
{    i++;
     j++;
}
else
{   i--;
    j--;
}
```

该程序段执行后的 i，j 值均为 1，若将 else 分句的大括号省掉，即：

```
int i=0,j=0;
if (i==j)
{    i++;
     j++;
}
else
  i--;
j--;
```

则程序段的执行结果就变为 i 值为 1，而 j 值仍为 0，从而可见大括号的重要性。顺便指出，if 后面的大括号绝对不能省掉，否则程序：



```
int i=0,j=0;
if (i==j)
    i++;
    j++;
else
{   i--;
    j--;
}
```

将出现编译错误：else 找不到配对的 if。

◉　分支结构嵌套

Java 语言允许对 if-else 条件语句进行嵌套使用。前述分支结构的语句部分，可以是任何语句(包括分支语句本身)，人们把分支结构的语句部分仍为分支结构的情况，称为分支结构嵌套。构造分支结构嵌套的主要目的是解决条件判断较多，较复杂的一些问题。常见的嵌套结构如下所示：




```
if (布尔表达式 1)
  if (布尔表达式 2)
        语句 1；
```

或

```
if (布尔表达式 1)
    语句 1；
else if (布尔表达式 2)
    语句 2；
else
   语句 3；
```

或

```
if (布尔表达式 1)
    if (布尔表达式 2)
      语句 1；
    else
      语句 2；
else
    语句 3；
```

当然，根据具体问题的不同，嵌套结构还可以设计成很多其他情形。

◉　switch 语句

上面已经介绍了单分支和双分支的选择结构，下面请看多分支结构，Java 语言多分支结构

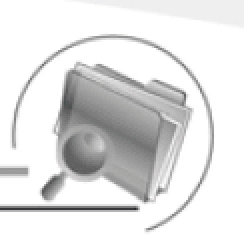
的实现语句是 switch，switch 语句的一般语法格式如下：

```
switch(表达式)
{case 判断值 1：语句 1；
 case 判断值 2：语句 2；
  . . .
 case 判断值 n：语句 n；
 [default：语句 n+1；]
}
```

其中，表达式的值必须为有序数值(如整型数或字符等)，不能为浮点数，而 case 语句中的判断值则须为常量值，有的教材称之为标号，代表一个 case 分支的入口，每一个 case 分支后面的语句可以是单条的，也可以是多条的，并且当有多条语句时，不需要加大括号{}将其括起来。default 子句是可选的，并且其位置必须在 switch 结构的末尾，当表达式的值与任何 case 常量值均不匹配时，就执行 default 子句，然后就退出 switch 结构了。若表达式的值与任何 case 常量值均不匹配，且无 default 子句，则程序不执行任何操作，直接跳出 switch 结构，继续后续的程序。下面请看一个例子。

【例 4-5】在控制台敲入 0 至 6 的数字，输出对应的星期数(0 对应星期天，1 对应星期一，依此类推)。

```
import java.io.* ;
class Test
{public static void main(String args[])throws IOException
  {int day;
   System.out.print("请输入星期数(0-6):") ;
   day=(int)(System.in.read())-'0';
   switch(day)
   {case 0: System.out.println(day +"表示是星期日");          break;
    case 1: System.out.println(day +"表示是星期一");          break;
    case 2: System.out.println(day +"表示是星期二");          break;
    case 3: System.out.println(day +"表示是星期三");          break;
    case 4: System.out.println(day +"表示是星期四");          break;
    case 5: System.out.println(day +"表示是星期五");          break;
    case 6: System.out.println(day +"表示是星期六");          break;
    default:  System.out.println(day+"是无效数!") ;
   }
  }
}
```



上例中，最后的 default 子句没有必要再添加 break 语句，因为它已经是 switch 结构的末尾了。一般情况下，switch 结构与 break 是相伴的，配套来使用。此外，使用 switch 结构时请注意以下问题：

(1) 允许多个不同的 case 标号执行相同的一段程序，比如以下情形：

```
    ...
case 常量 i:
case 常量 j:
        语句;
        break;
    ...
```

(2) 每一个 case 子句的常量值必须各不相同。

最后，需要指出的是，switch 结构的程序通常也可以用 if-else 语句来实现，读者可以试着将上述程序改写成 if-else 的结构，并对比一下二者差别。但反过来，if-else 的结构则并不总能由 switch 结构来实现。用 switch 结构编写的程序通常显得更简练些，可读性也更好，程序执行效率也更高。因此，请读者在设计程序时注意不同的分支结构的选用。下面继续介绍程序的第 3 种流程结构：循环。



4.5.2 循环结构

在进行程序设计时，经常会碰到一些计算并不很复杂，但却要重复进行相同的处理操作的问题。比如：

(1) 计算累加和 1+2+3+…+100。

(2) 计算阶乘，如 10！。

(3) 计算一笔钱在银行存了若干年后，连本带息有多少？

由于上述问题本身的特点，导致目前为止所介绍的语句都无法表示这种结构。如问题(1)，用一条语句：sum = 1+2+3+…+100 来求解，则赋值表达式太长了，改成多条赋值语句：sum +=1; sum +=2; sum +=3; …; sum +=100;也不行，即便加到 100 那也有 100 条语句，程序过于臃肿，不利编辑、存储和运行。因此，在 Java 语言中，引入了 3 种语句：while、do-while 以及 for 来解决这类问题。人们把这类问题的结构称为循环结构，把这 3 种实现语句称为循环语句。

◉ while 语句

while 语句的一般语法格式如下：

```
while(条件表达式)
{  循环体; }
```

while 是系统关键字，首先计算条件表达式的布尔值，若为 true 则执行循环体，然后再计

算条件表达式的布尔值，只要是 true 就循环往复执行下去，直到布尔值为 false 时才结束退出 while 结构。其中循环体可以是复合语句、简单语句甚至是空语句，但一般情况下，循环体中应包含有能修改条件表达式取值的语句，否则就容易出现“死循环”(程序毫无意义地无限循环下去)。例如：while(1)；这里，循环体为一空语句，而条件表达式为一常量 1(Java 语言里，0 代表 false，非 0 为 true)，因此这是一死循环。

【例 4-6】利用 while 语句实现 1 到 100 的累加。

```
public class Test
{
     public static void main(String[] args)
    {
          int sum=0;   //累加和变量 sum
          int i=1;          //  控制变量 i
          while(i<=100)
          {
             sum+=i;
             i++;
          }
        System.out.println("累加和为："+sum);
     }
}
```

该程序有几点需要注意：

(1) 存放累加和的变量初始值一般赋值为 0。

(2) 变量 i 既是累加数，同时又是控制变量(控制循环体的循环次数)。

(3) 循环体语句　　sum+=i;
　　　　　　　　　i++;

可以合并简写为：　sum+=i++;

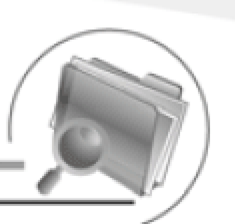
(4) while 循环体语句多于一条，因而必须以复合语句形式出现，千万别漏了大括号。

下面再来看阶乘的例子。

【例 4-7】利用 while 语句求 10 的阶乘。

```
public class Test
{public static void main(String[] args)
  {
      long jc=1;
      int i=1;
      while(i<=10)
      {
          jc*=i;
```





```
            i++;
        }
        System.out.println((i-1)+"!结果："+jc);
    }
}
```

本程序需要注意的要点有：

(1) 求阶乘的积时，变量 jc 初始值应为 1。

(2) 由于阶乘的积，数值往往比较大，因此要注意防止溢出，比如尽量选用取值范围大的长整型 long。

【例 4-8】 有一条长的阶梯，如果每步 2 阶，则最后剩 1 阶，每步 3 阶则剩 2 阶，每步 5 阶则剩 4 阶，每步 6 阶则剩 5 阶，只有每步 7 阶的最后才刚好走完，一阶不剩，问这条阶梯最少共有多少阶？

```
public class Test
{public static void main(String[] args)
  {
      int i=1;
      while(!(i%2==1&&i%3==2&&i%5==4&&i%6==5&&i%7==0))
      {
          i++;
      }
      System.out.println("这条阶梯最少有："+i+"阶");
  }
}
```



程序运行结果如下：

```
这条阶梯最少有：119 阶
```

该程序的关键是 while 结构的条件表达式要写对。其实满足题目要求的阶梯数有无限多个，119 阶只是其中最小的一个，假如现在想知道在 1 万个阶梯内，都有哪些阶梯数满足题意的话，可以这样改写程序中的 while 结构：

```
while(i<=10000)
{
 if(i%2==1&&i%3==2&&i%5==4&&i%6==5&&i%7==0)
    System.out.print(i+"阶  ");
  i++;
}
```

新程序运行结果如下：

119 阶 329 阶 539 阶 749 阶 959 阶 1169 阶 1379 阶 1589 阶 1799 阶 2009 阶 2219 阶 2429 阶 2639 阶 2849 阶 3059 阶 3269 阶 3479 阶 3689 阶 3899 阶 4109 阶 4319 阶 4529 阶 4739 阶 4949 阶 5159 阶 5369 阶 5579 阶 5789 阶 5999 阶 6209 阶 6419 阶 6629 阶 6839 阶 7049 阶 7259 阶 7469 阶 7679 阶 7889 阶 8099 阶 8309 阶 8519 阶 8729 阶 8939 阶 9149 阶 9359 阶 9569 阶 9779 阶 9989 阶

利用计算机计算这个问题时，所需时间非常之短，若让人手工去计算这个问题，即便是世界上反应最快的人，也是需要不少时间的。假如不是求 1 万个阶梯内的，而是求 100 万个呢，到底这之间又会有多少种阶梯数能满足题意，有兴趣的读者可以自行编写程序，并上机尝试一下。

⦿ do-while 语句

do-while 语句的语法格式如下：

```
do
{
  循环体;
}while(条件表达式);
```

首先先执行循环体一遍，然后再判断条件表达式的取值，若为 true 则返回继续执行循环体，直至条件表达式的取值变为 false。

do-while 语句与 while 语句结构比较接近，通常情况下，它们之间可以互相转换。比如将【例 4-6】改写成用 do-while 语句来实现，修改代码如下：

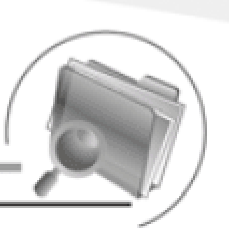
```
do
{
    sum+=i;
    i++;
} while(i<=100);
```

本例修改时，常犯的错误是：在 while 判断后漏掉了分号(;)，而这在前面 while 结构中则是没有的。下面再举个利用 do-while 实现循环结构的例子。

【例 4-9】假定在 Bank 中存款额 5000 元，按 6.25%的年利率计算，试问过多少年后就会连本带利翻一翻？试编程实现之。

```
public class Test
{public static void main(String[] args)
  {
    double m=5000.0; //初始存款额
    double s=m;              // 当前存款额
    int count=0;             // 存款年数
    do
```



```
        {
            s=(1+0.0625)*s;
            count++;
        }while(s<2*m);
        System.out.println(count+"年后连本带利翻一翻！");
    }
}
```

程序运行结果如下：

```
12 年后连本带利翻一翻！
```

本例中，定义整型变量 count 作为计数器，用来记录存款年数。事实上，在很多应用中，都需要用到这种看似简单，却很有用的计数器，因此，大家(尤其是初学者)要注意学习模仿。记得曾经有人说过这么一句话：好程序都是模仿出来的！这话是否绝对正确，其实并不重要，重要的是它告诉人们一条学习编程的途径——模仿，笔者的个人体会是，多参考模仿好的程序，如一些著名软件公司所提供的源代码或者编译系统自带的库函数(方法)代码等。



虽然 do-while 语句与 while 语句结构比较接近，但有一点需要特别注意的是：while 语句的循环体有可能一次也不被执行，而 do-while 语句的循环体至少被执行一次，这是二者最大的区别。

◉ for 语句

for 语句的一般语法格式如下：

```
for(表达式 1；条件表达式 2；表达式 3)
  {  循环体；  }
```

每个 for 语句都有一个用于控制循环开始和结束的变量，即循环控制变量。表达式 1 一般用来给循环控制变量赋初值，它仅在刚开始时被执行一次，以后就不再被执行。表达式 2 是一个条件表达式，根据其取值的不同，决定循环体是否被执行，若为 true，则执行循环体，然后再执行表达式 3，这个表达式 3 通常用作修改循环控制变量之用，以避免陷入死循环，接着又判断条件表达式 2 的布尔值，若还为 true，则继续上述循环，直至布尔值变为 false。

for 语句是 Java 语言 3 种循环语句中功能较强，使用也较广泛的一种。现举例如下。

【例 4-10】利用 for 语句实现 1 到 100 的累加。

```
public class Test
{
    public static void main(String[] args)
    {
        int sum=0;                    // 累加和变量 sum
        for(int i=1; i<=100;i++)    // 控制变量 i
        {
```

```
            sum+=i;
         }
        System.out.println("累加和为："+sum);
     }
}
```

程序运行结果如下：

```
累加和为：5050
```

【例 4-11】假定在 Bank 中存款额 5000 元，按 6.25%的年利率计算，试问过多少年后就会连本带利翻一翻？试用 for 语句编程实现之。

```
public class Test
{public static void main(String[] args)
   {
      double m=5000.0;           // 初始存款额
      double s=m;                // 当前存款额
      int count=0;               // 存款年数
      for(;s<2*m;s=(1+0.0625)*s)
          count++;
       System.out.println(count+"年后连本带利翻一翻！");
   }
}
```

本例将 for 结构中赋初值的表达式 1 拿到上面去了，这是允许的。甚至还可以对 for 语句改写为：

```
for(;s<2*m;)
{    count++;
     s=(1+0.0625)*s
}
```

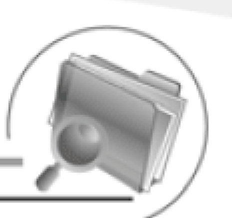
此时，for 结构的表达式 1 和表达式 3 均为空。其实不管怎么改写，只要程序遵循 for 语句的执行流程执行后能得出正确结果即可。

4.5.3 跳转语句

Java 中的跳转语句包括 break 语句和 continue 语句。

◉ break 语句

break 语句的作用是使程序的流程从一个语句块的内部跳转出来，如前述的 switch 结构以



及循环结构。break 语句的语法格式为：

```
break  [标号];
```

其中的标号是可选的，如前面介绍的 switch 结构程序就没有使用标号。不使用标号的 break 语句只能跳出当前的 switch 或循环结构，而带标号的则可以跳出由标号指出的语句块，并从语句块的下条语句处继续程序的执行。因此，带标号的 break 语句可以用来跳出多重循环结构。下面分别举例说明。

【例 4-12】写出以下程序执行后的输出结果。

```
public class Test
{public static void main(String[] args)
 {
        int i ,s=0;
        for(i=1;i<=100;i++)
        {
          s+=i;
          if(s>50)
            break;
        }
        System.out.println("s="+s);
 }
}
```

程序运行结果如下：

```
s=55
```

【例 4-13】写出以下程序执行后的输出结果。

```
public class Test
{      public static void main(String[] args)
       {      int   s=0,i=1;
              label:
              while(true)
              {      while(true)
                        {   if (i%2==0)
                                 break ;          // 不带标号
                             if(s>50)
                                 break label;     // 带标号
                             s+=i++;
                        }
                     i++;
```



计算机基础与实训教材系列

```
            }
            System.out.println("s="+s);
        }
}
```

程序运行结果如下：

```
s=64
```

分析以上程序，发现执行过程为：将 1、3、5 等奇数累加到 s 变量，直到 s 值超出 50。不带标号的 break 语句用来跳出内层 while 循环，以避开对偶数的累加，带标号的 break 语句用来跳出 label 所标识的两重 while 循环结构，然后执行输出语句，显示当前 s 变量的值。这里需要指出的是，若没有带标号的 break 语句，则两重无限循环结构就会变成死循环。在一些特殊情况，带标号的 break 语句确有妙用，但对于一般的程序设计，请慎用，千万不要滥用。



◉ continue

continue 语句只能用于循环结构，它也有两种使用形式：不带标号和带标号。前者的功能是提前结束本次循环，即跳过当前循环体的其他后续语句，提前进入下一轮循环体继续执行。对于 while 和 do-while 循环，不带标号的 continue 语句会使流程直接跳转到条件表达式，而对于 for 循环，则跳转至表达式 3，修改控制变量后再进行条件表达式 2 的判断。带标号 continue 语句多用在多重循环结构中，标号的位置与 break 语句的标号位置相类似，一般需放至整个循环结构的前面，用来标识这个循环结构，一旦内层循环执行了带标号 continue 语句，程序流程则跳转到标号处的最外层循环，具体是：while 和 do-while 循环，跳转到条件表达式，for 循环，跳转至表达式 3。下面分别举例予以说明。

【例 4-14】写出以下程序执行后的输出结果。

```
public class Test
{     public static void main(String[] args)
        {     int   s=0,i=0;
              do
              {  i++;
                 if (i%2!=0)
                     continue;
                  s+=i;
              }while(s<50);
            System.out.println("s="+s);
        }
}
```

程序运行结果如下：

```
s=56
```



上述程序 do-while 循环用来计算偶数 2、4、6…的累加和，条件是和小于 50，最后退出循环结构时，累加和为 56。其中的 continue 语句作用是当遇到奇数时，则跳过，不予累加。程序中 i++语句与 if 条件语句的位置不能对调，对调则会使程序陷入死循环，请读者自行分析一下。

【例 4-15】写出以下程序执行后的输出结果。

```
public class Test
{      public static void main(String[] args)
       {    int i,j;
            label:
            for(i=1;i<=200;i++)           // 查找 1 到 200 以内的素数
            {     for(j=2;j<i;j++)         // 检验是否不满足素数条件
                       if (i%j==0)         // 不满足
                            continue label; // 跳过后面不必要的检验
                  System.out.print(" "+i); // 打印素数
            }
       }
}
```

程序运行结果如下：

```
1 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199
```

当内层循环检验到 if 的条件表达式 i%j==0 为 true 时，即除了 1 和自身外，i 还能被其他的整数整除，因而 i 肯定不是素数，这时就没有必要继续循环判断下去，故通过 continue label;语句将程序流程跳转至外层循环的表达式 3(i++)处，继续下一个数的判断工作。



4.6　使用 Eclipse 开发 Java 程序

通过前面的学习，读者已经了解了 Java 的一些基本语法，下面介绍如何使用 Eclipse 来开发 Java 程序。还是从一个简单的排序程序 Sorter.java 开始，程序功能非常简单，就是将一个词组数组的内容通过系统提供的默认方法(区分大小写，小写字母大于大写字母)和不区分大小的方法进行排序。程序代码清单如下：

```
//程序清单 Sorter.java：
import java.text.*;
import java.util.*;
public class Sorter {
  public static class CaseInsensitiveComparator
      implements Comparator {
```

```
    public int compare(Object element1, Object element2) {
      String lower1 = element1.toString().toLowerCase();
      String lower2 = element2.toString().toLowerCase();
      return lower1.compareTo(lower2);
    }
  }
  public static void main(String args[]) {
    String words[] =
      {"JSP", "jsp", "Java", "JAVA", "Servlet", "servlet", "J2EE"};
      List list = new ArrayList(Arrays.asList(words));
    System.out.println("原始列表:");
    System.out.println(list);
      //默认排序方式
    Collections.sort(list);
    System.out.println("默认的排序结果:");
    System.out.println(list);
      //不区分大小写的排序方式
    Comparator comp = new CaseInsensitiveComparator();
    Collections.sort(list, comp);
    System.out.println("不区分大小写的排序结果:");
    System.out.println(list);
  }
}
```



Java 应用程序(可执行)的入口点是 static main 方法，它必须包含在一个类中(将在第 5 章介绍类的用法，这里先简要介绍一下代码的功能)。

首先程序定义了一个 String 类型的数组 words 来存储原始数据，接着通过 ArrayList 类的构造方法(第 5 章详细介绍)将该数组转化为一个列表 List。然后在系统输出中输出显示列表内容。

接着使用系统提供的 Collections.sort 方法将该列表重新排序，然后再次输出显示。最后，使用不区分大小写的比较方法进行新的排序，并输出显示最终结果。

具体的系统类和它们的方法可以查看 JDK 的帮助文档。

下面就用 Eclipse 来开发这个简单的程序：

(1) 双击 Eclipse.exe 文件。

(2) 在打开的 Eclipse 平台中，选择【File】|【New】|【Project】命令，打开【New Project】(新建项目)对话框，如图 4-1 所示。

(3) 选择【Java Project】选项，单击 Next 按钮，弹出【New Java Project】对话框，如图 4-2 所示。

(4) 在【Project name】文本框中输入项目名称，如 MyJava，单击【Finish】按钮完成项目设置。Eclipse 会自动转为 Java 视图显示，在左边的【Package Explorer】窗口中显示新建的 MyJava



项目内容，如图 4-3 所示。

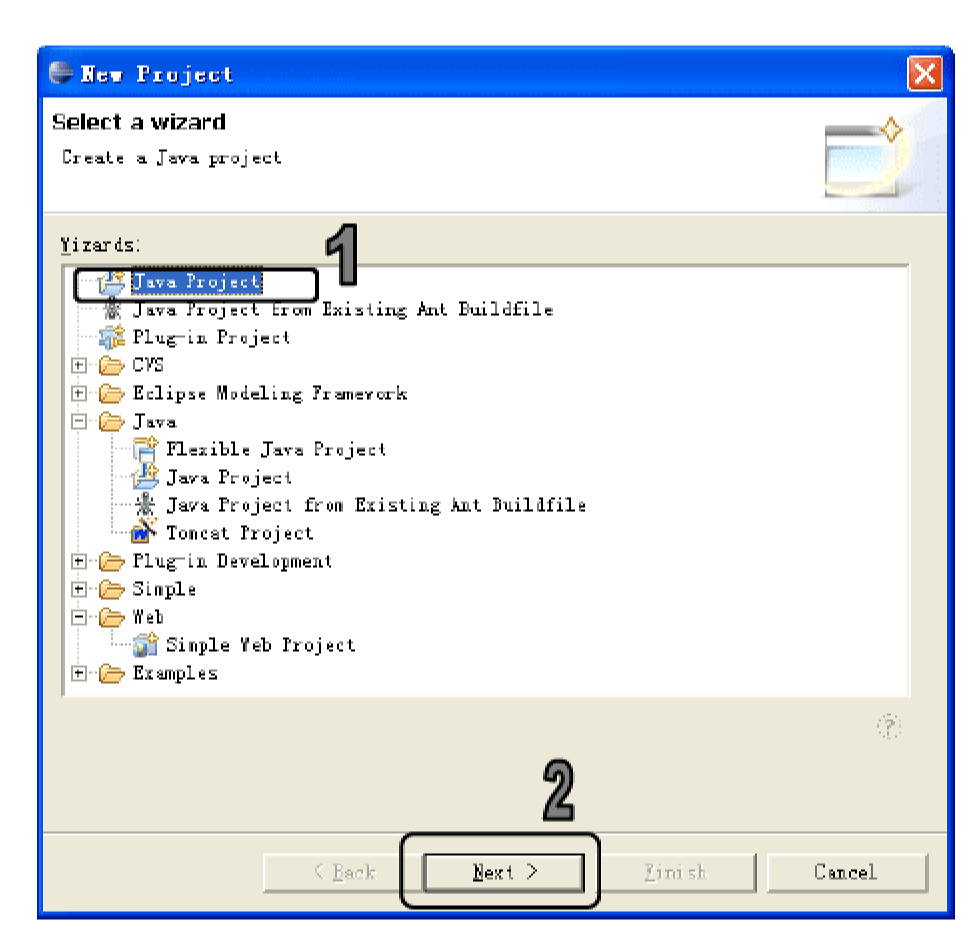


图 4-1　新建项目对话框

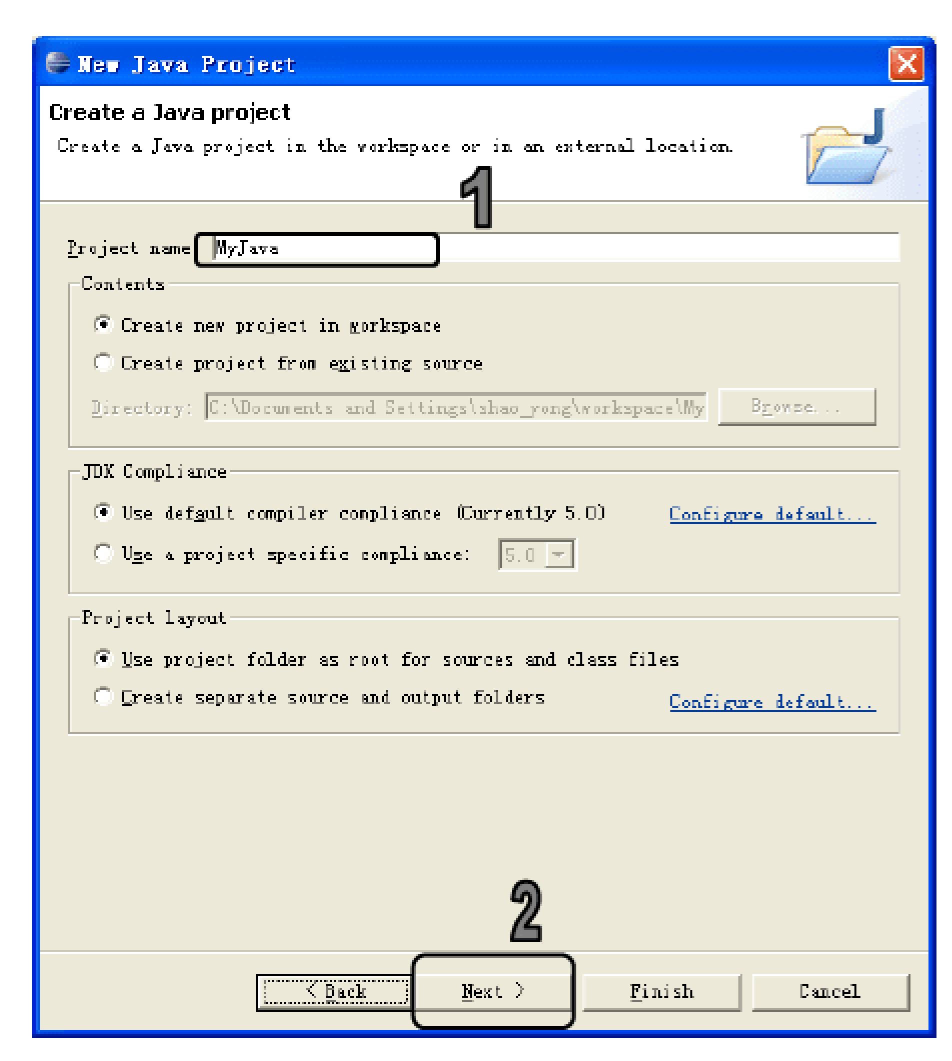


图 4-2　新建 Java 项目对话框

(5) 选择【File】|【New】|【Class】命令新建一个 Java 类，弹出如图 4-4 所示的 New Java Class 对话框。

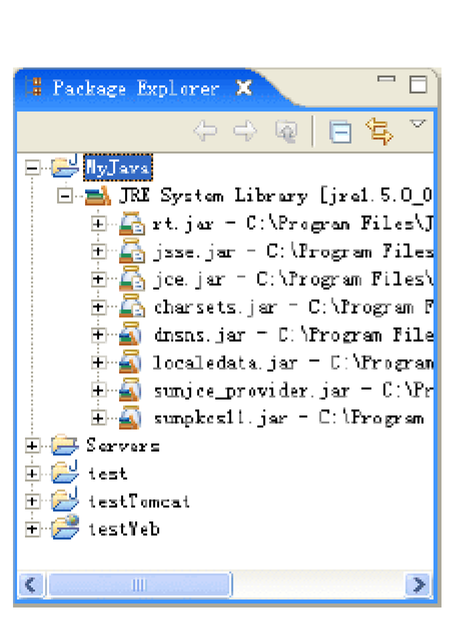


图 4-3　Package Explorer

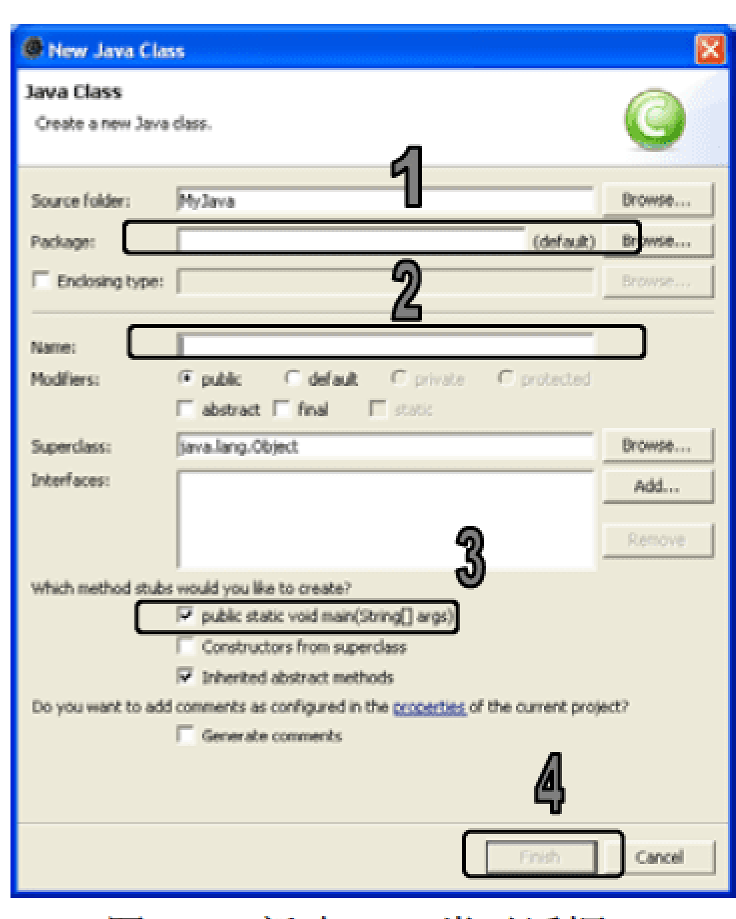


图 4-4　新建 Java 类对话框

(6) 在【Package】文本框中输入需要的包名或使用默认包(空)，在【Name】文本框中输入 Sorter，选中【public static void main(String[] args)】复选框。单击【Finish】按钮将创建一个名为 Sorter 的 Java 类(注意：在 Java 中，文件名要与文件中的 public 类名字一致，否则会出错)。Eclipse 会通过 Java 编辑器打开新建的 Sorter.java 文件，输入程序代码，如图 4-5 所示。

(7) 选择【Run】|【Run As】|【Java Application】命令，Eclipse 将编译并执行 Sorter 程序，同时在 Console 中打印出相应的结果，如图 4-6 所示。



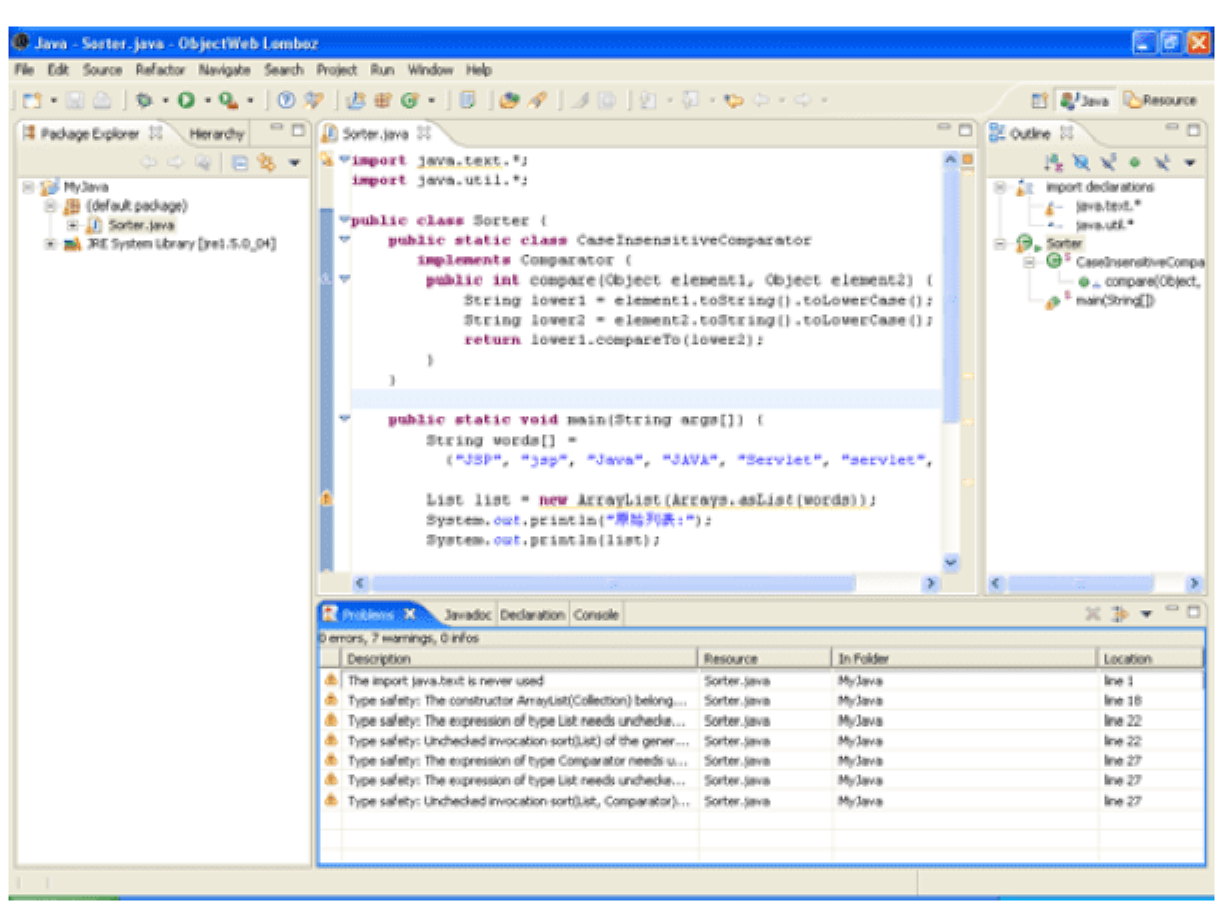


图 4-5　编写 Sorter 程序



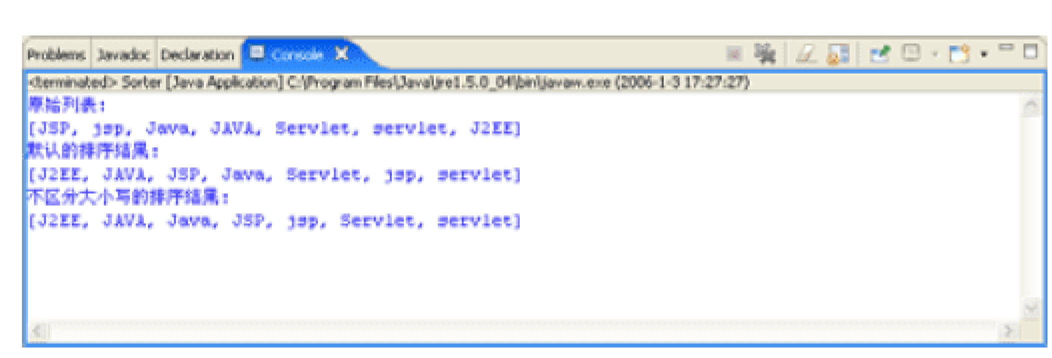


图 4-6　Sorter 程序输出结果

4.7　上机练习

本章上机实验主要练习如何创建 Java 项目，如何编写、编译并运行 Java 程序。这里创建一个名为 Test.java 的 Java 程序，用来测试本章介绍到的各种 Java 语法。

(1) 使用资源管理器打开 Eclipse 所在的文件夹，双击 Eclipse.exe 文件，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建项目。

(3) 在打开的【New Project】对话框中，选择【Java Project】选项，单击【Next】按钮打开【New Java Project】对话框。

(4) 在【New Java Project】对话框的【Project Name】文本框中输入项目名称，如 MyJava，单击【Finish】按钮后 Eclipse 将自动创建一个 Java 项目。

(5) 在 Eclipse 主窗口左侧的【Package Explorer】窗口中将会出现新建的 MyJava 项目。

(6) 选择【File】|【New】|【Class】命令弹出【New Java Class】对话框建立 Java 类。

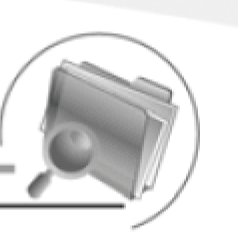
(7) 在【New Java Class】对话框的【Package】文本框中输入 Java 类所使用的包名，例如 ch4。

(8) 在【Name】文本框中输入 Java 类的名字，如 Test。

(9) 选中 public static void main(String[] args)复选框，单击 Finish 按钮完成 Java 类的创建。

(10) Eclipse 创建的新类，包括了基本的类框架。在类的编辑窗口中输入 Test.Java 程序代码。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用



红色显示。将鼠标移动到错误位置，会显示相应的错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 在 Package Explorer 窗口中右键单击 Test.java 类，从弹出的菜单中选择【Run As】命令，如果 Java 类语法正确，而且具有合法的 main 方法，则在【Run As】命令的子菜单中会出现【Java Application】菜单命令。

(14) 选择【Run As】|【Java Application】命令，将在 Console 中显示打印信息。通过修改代码重复步骤(11)~(14)，练习各种语法的使用。

4.8 习题

4.8.1 填空题



1. Java 中的数据类型包括两种类型：____________和____________。
2. Java 的注释语句有 3 种类型：____________、____________和____________。
3. 表达式包括：____________、____________、____________和____________。
4. 循环语句包括：____________、____________和____________。

4.8.2 选择题

1. 下面哪种数据类型是基本类型(　　)。

A. class　　B. interface　　C. 数组　　D. char

2. 下面哪种数据类型是引用类型(　　)。

A. boolean　　B. int　　C. interface　　D. long

3. 下面哪一个运算符具有 3 个操作数(　　)。

A. *　　B. ++　　C. &&　　D. ? :

4. 下面哪个运算符只有一个操作数(　　)。

A. ~　　B. ||　　C. <<　　D. /

5. 下面哪个语句是条件语句(　　)。

A. break 语句　　B. while 语句　　C. if 语句　　D. for 语句

4.8.3 问答题

1. 下面的代码片断是否有误？如果有，请指出并修改。

```
switch (i) {
  case 1:
    System.out.println("1");
  case 2:
    System.out.println("2");
}
i=10;
do {
    System.out.println("i<10");
    i=i+1
} while (i<10)
```

2. 下述程序的输出结果是多少？

```
public class Test
{public static void main(String[] args)
  {
        int i ,s=0;
        for(i=1;i<=100;i++)
        {
          if(i%3==0)
            continue;
          s+=i;
        }
        System.out.println("s="+s);
  }
}
```



3. 下述程序的输出结果是多少？

```
public class Test
{public static void main(String[] args)
  {
        int i ,s=0;
        for(i=1;i<=100;i++)
        {
          s+=i;
          if(s>100)
            break;
        }
        System.out.println("s="+s);
  }
}
```



Java 面向对象编程

学习目标

要深入地学习 Java/JSP，还必须了解并掌握面向对象的编程技术，面向对象编程的英文全称是 Object-Oriented Programming，简称 OOP。OOP 是一种非常重要的编程思想，是深入掌握 Java 的必要条件，是迈向高手境界的必经之路。理解了 OOP 技术之后，很多以前困惑的问题也会迎刃而解。深入 JSP 和 Java 的世界，成为一个真正的 Web 开发人员，就应该对这些细节深入学习并掌握它们。

本章重点

- 类和对象
- 访问控制符
- 继承与多态

5.1 类

Java 语言作为一种面向对象的程序设计语言，用它进行程序设计必须将自己的思想转入一个面向对象的世界，以对象世界的思维方式来思考问题。编写一个 Java 程序，在某种程度上就是在定义类和创建类对象。定义类和建立对象是 Java 编程的主要任务。抽象和封装这两个面向对象程序设计的重要特点主要体现在类的定义及使用上。类是组成 Java 程序的基本要素，它封装了一类对象的状态和方法，定义一个新类，就是创建一个新的“数据类型”。

Java 的类分为两大部分：系统定义的类和用户自定义的类。Java 的类库是系统定义的类，它是系统提供的已实现的标准类的集合，提供了 Java 程序与运行它的系统软件之间的接口。Java 类库是一组由其他开发人员或软件供应商编写好的 Java 程序模块，每个模块通常对应一种特定

的基本功能和任务，这样当自己编写的 Java 程序需要完成其中某一功能的时候，就可以直接利用这些现成的类库，而不需要一切从头编写。Java 的类库大部分是由 SUN 公司提供的。这些类库称为基础类库。

本节主要介绍如何创建用户自定义类。

【例 5-1】自定义【教师】类。

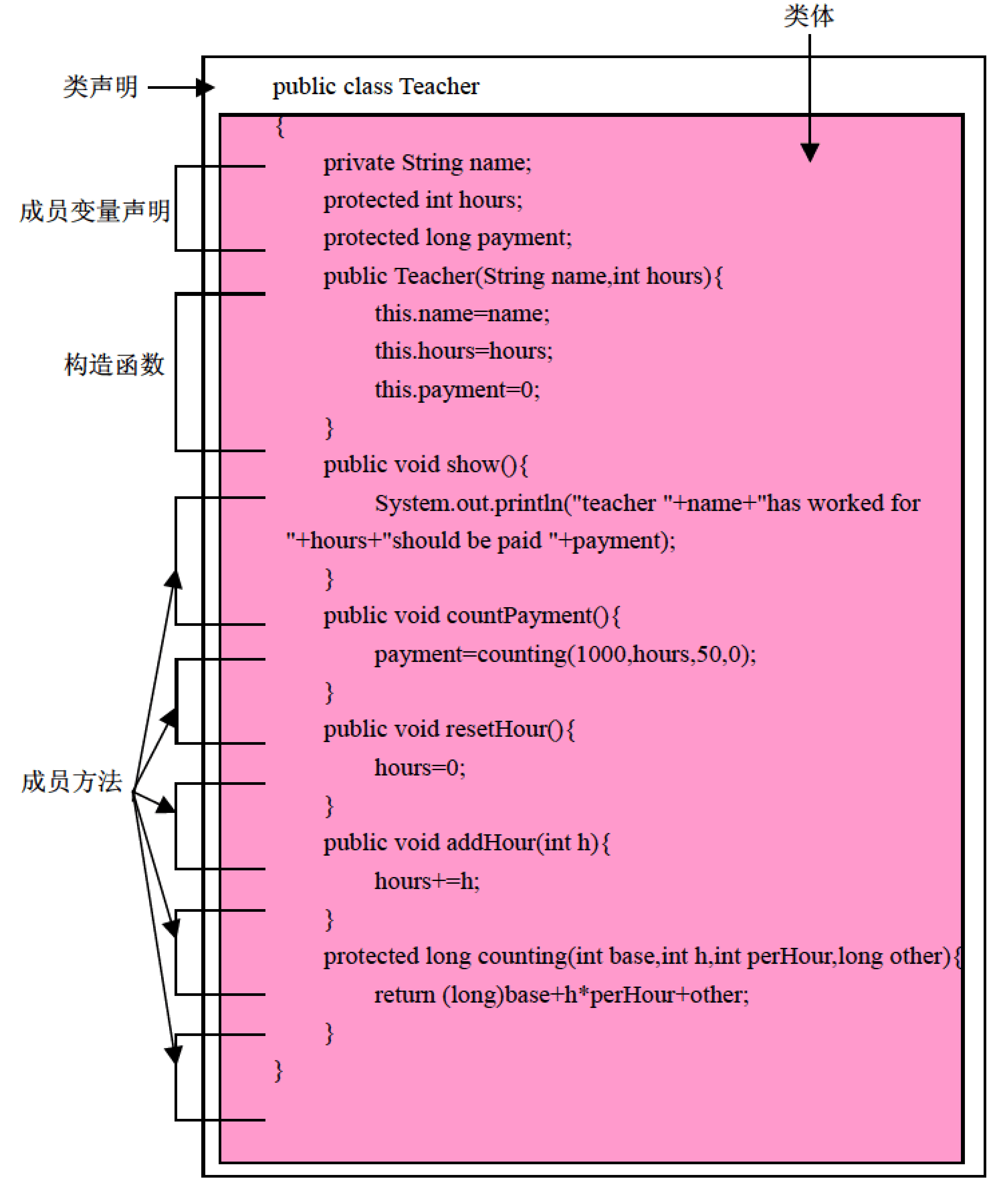


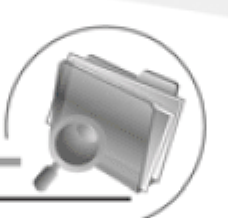
图 5-1　Teacher 类以及代码结构

图 5-1 给出了组成类的两个主要部分：类声明和类体，并给出了实现代码。下面分别对每部分做一介绍。

5.1.1　类声明

类声明格式如下：





```
[类的修饰字]class 类名[extends 父类名][implements 接口列表]
{
    ……  //类体
}
```

这里 class 是声明一个类的关键字，类名是要声明的类的名字，它必须是一个合法的 Java 标识符。根据声明类的需要，类声明还可以包含 3 个选项：

- 声明类的修饰符。
- 说明该类的父类。
- 说明该类所实现的接口。

其中 class 关键字是必须的，所有其他的部分都是可选的。下面对类声明的 3 个选项给出更详细的介绍。

(1) 类修饰符

类修饰符说明这个类是一个什么样的类。类修饰符可以是 public、abstract 或 final，如果没有声明这些可选的类修饰符，Java 编译器将给出默认值。类修饰符的含义分别如下。

public：这个 public 关键字声明了类可以在其他任何的类中使用。缺省时，该类只能被同一个包中的其他类使用。

abstract：声明这个类不能被实例化，即这个类是抽象类。

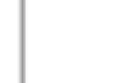
final：声明了类不能被继承，即没有子类。

(2) 说明类的父类

在 Java 中，除 Object 之外，每个类都有一个父类。Object 是 Java 语言中唯一没有父类的类，如果某个类没有指明父类，Java 就认为它是 Object 的子类。因此，所有其他类都是 Object 的直接子类或间接子类。值得注意的是，在 extends 之后只能跟唯一的父类名，即使用 extends 只能实现单继承。

(3) 说明一个类所实现的接口

为了声明一个类要实现的一个或多个接口，可以使用关键字 implements，并且在其后面给出由该类实现的接口的名字表，它们是以逗号分隔的。接口的定义和实现将在后面具体介绍。



5.1.2　类体

类体中定义了该类中所有的变量和该类所支持的方法。通常变量在方法前定义(并不一定要求)，类体定义如下：

```
class className{          //类声明
[public|protected|private][static][final][transient][volatile]
            type variableName;              //成员变量
[public|protected|private][static][final|abstract][native][synchronized]
             return TypemethodName([paramList])[throws exceptionList]
```




```
        {statements}          //成员方法
}
```

类中所定义的变量和方法都是类的成员。对类的成员可以设定访问权限，来限定其他对象对它的访问，访问权限可以有以下几种：public、protected、private、default，这些将在后面详细讨论。同时，对类的成员来说，又可以分为实例成员和类成员两种，将在后面详细讨论。

5.1.3 成员变量

最简单的成员变量的声明方式如下：

```
type 成员变量名;
```

这里的 type 可以是 Java 中任意的数据结构，包括简单类型、类、接口、数组。在一个类中的成员变量应该是唯一的。



类的成员变量和在方法中所声明的局部变量是不同的，成员变量的作用域是整个类，而局部变量的作用域只是方法内部。对一个成员变量，可以用以下的修饰符限定。

(1) static：用来指示一个变量是静态变量(类变量)，不需要实例化该类即可使用。所有该类的对象都使用同一个类变量。没有用 static 修饰的变量是实例变量，必须实例化该类才可以使用实例变量。该类的不同对象都各自拥有自身的实例变量的版本。类方法通常只能使用类变量，不能使用实例变量。

(2) final：用来声明一个常量，如：

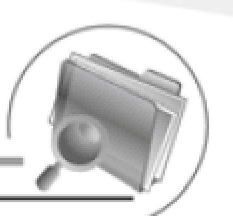
```
classFinalVar{
    final int CONSTANT=50;
    ……
}
```

此例中声明了常量 CONSTANT，并赋值为 50。对于用 final 限定的常量，在程序中不能改变它的值。通常常量名用大写字母表示。

(3) transient：用来声明一个暂时性变量，如：

```
class TransientVar{
    transient TransientV;
    ……
}
```

在默认情况下，类中所有变量都是对象永久状态的一部分，当对象被存档时(串行化)，这些变量必须同时被保存。用 transient 限定的变量则指示 Java 虚拟机，该变量并不属于对象的永久状态。它主要用于实现不同对象的存档功能。



(4) volatile：声明一个共享变量，如下所示。

```
class VolatileVar{
        volatile int volatileV;
        ……
}
```

由多个并发线程共享的变量可以用 volatile 来修饰，使得各个线程对该变量的访问能保持一致。

5.1.4 成员方法

方法的实现包括两部分内容：方法声明和方法体，如下：

```
[public|protected|private][static][final|abstract][native][synchronized]
return TypemethodName([paramList])[throws exceptionList]
{statements}
```



1. 方法声明

最简单的方法声明包括方法名和返回类型，如下：

```
returnType methodName( ){
        …… //方法体
}
```

其中返回类型可以是任意的 Java 数据类型，当一个方法不需要返回值时，则必须声明其返回类型为 void。

(1) 方法的参数

在很多方法的声明中，都要给出一些外部参数，为方法的实现提供信息。这是在声明一个方法时，通过列出它的参数表来完成的。参数表指明每个参数的名字和类型，各参数之间用逗号分隔，如下：

```
returnType methodName(type name[, type name[, …]]){
    ……
}
```

对于类中的方法，与成员变量相同，可以限定它的访问权限：

- static 限定它为类方法。
- abstract 或 final 指明方法是否可被重写。
- native 用来把 Java 代码和其他语言的代码集成起来。

⊙ synchronized 用来控制多个并发线程对共享数据的访问。

⊙ throws ExceptionList 用来处理例外。

【例 5-2】方法中的参数。

```
class Circle{
    int x,y,radius;                          // x,y,radius 是成员变量
    public Circle(int x,int y,int radius){   // x,y,radius 是参数
    ……
    }
}
```

Circle 类有 3 个成员变量：x、y 和 radius。在 Circle 类的构造函数中有 3 个参数，名字也是 x、y 和 radius。在方法中出现的 x、y 和 radius 指的是参数名，而不是成员变量名。如果要访问这些同名的成员变量，必须通过【当前对象】指示符 this 来引用它。例如：

```
class Circle{
    int x,y,radius;
    public Circle(int x,int y,int radius){
      this.x=x;
      this.y=y;
      this.radius=radius;
    }
}
```

带 this 前缀的变量为成员变量，这样，参数和成员变量便一目了然了。this 表示的是当前对象本身，更准确地说，this 代表了当前对象的一个引用。对象的引用可以理解为对象的另一个名字，通过引用可以顺利地访问到该对象，包括访问、修改对象的成员变量、调用对象的方法。

(2) 方法的参数传递

在 Java 中，可以把任何有效数据类型的参数传递到方法中，这些类型必须预先定义好。

另外，参数的类型可以是简单数据类型，也可以是引用数据类型(数组类型，类或接口)。对于简单数据类型，Java 实现的是值传送，方法接收的是参数的值，但并不能改变这些参数的值，如果要改变参数的值，则要用到引用数据类型，因为引用数据类型传递给方法的是数据在内存中的地址，方法中对数据的操作可以改变数据的值。【例 5-3】说明了简单数据类型与引用数据类型的区别。

【例 5-3】方法中简单数据类型和引用数据类型的区别。

```
public class PassTest{
  float ptValue;
  public static void main(String args[]){
    int val;
    PassTest pt=new PassTest( );      //生成一个类的实例 pt
```



```
    val=11;
    System.out.println("Original Int Value is:"+val);
    pt.changeInt(val);           //简单数据类型
    System.out.println("Int Value after change is:"+val);
    pt.ptValue=101f;
    System.out.println("Original ptValue Value is:"+pt.ptValue);
    pt.changeObjValue(pt);     //引用数据类型
    System.out.println("ptValue after change is:"+pt.Value);
    }
    public void changeInt(int value){
    value=55;
    }
    public void changeObjValue(PassTest ref){
    ref.ptValue=99f;
    }
}
```

运行结果如下：

```
c:\java PassTest
Original Int Value is:11
Int Value after change is:55
Original ptValue Value is:101.0
ptValue after change is:99.0
```

在类 PassTest 中定义了两个方法：changeInt(int value)和 changeObjValue(PassTest ref)，changeInt(int value)接收的参数是 int 类型的值，方法内部对接收到的 value 值进行了重新赋值，但由于该方法接收的是值参数，所以方法内进行的 value 值的修改不影响方法外的 value 值。而 changeObjValue(PassTest ref)接收的参数值是引用类型的，所以在该方法中对引用参数所指的对象的成员方法进行修改，是对该对象所占实际内存空间的修改，经过该方法作用之后，pt.Value 的值发生了真实的变化。

2. 方法体

方法体是对方法的实现。它包括局部变量的声明以及所有合法的 Java 指令。方法体中可以声明该方法中所用到的局部变量，它的作用域只在该方法内部，当方法返回时，局部变量也不再存在。如果局部变量的名字和类的成员变量的名字相同，则类的成员变量被隐藏。

【例 5-4】成员变量和局部变量的作用域示例。

```
class Variable{
    int x=0,y=0,z=0;   //类的成员变量
    void init(int x,int y){
```



```
        this.x=x;
        this.y=y;
        int z=5;        //局部变量
        System.out.println("****in init****");
        System.out.println("x="+x+"    y="+y+"    z="+z);
    }
}

public class VariableTest{
    public static void main(String args[]){
        Variable v=new Variable();
        System.out.println("****before init****");
        System.out.println("x="+v.x+"    y="+v.y+"    z="+v.z);
        v.init(20,30);
        System.out.println("****after init****");
        System.out.println("x="+v.x+"    y="+v.y+"    z="+v.z);
    }
}
```

运行结果是：

```
C:\>java VariableTest
****before init****
x=0    y=0    z=0
****in init****
x=20    y=30    z=5
****after init****
x=20    y=30    z=0
```

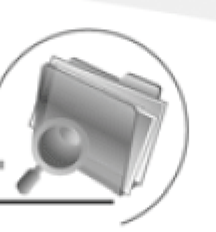
从【例 5-4】中可以看到局部变量 z 和类的成员变量 z 的作用域是不同的。

5.1.5　方法重载

方法重载即指多个方法可以享有相同的名字。但是这些方法的参数必须不同，或者是参数个数不同，或者是参数类型不同。以下的【例 5-5】中通过方法重载分别接收一个或几个不同数据类型的数据。

【例 5-5】方法重载应用举例。

```
class MethodOverloading{
    void receive(int i){
```



```
            System.out.println("Receive one int data");
            System.out.println("i＝"+i);
        }
        void receive(int x，int y){
            System.out.println("Receive two int datum");
            System.out.println("x="+x+" y="+y);
        }
        void receive(double d){
            System.out.println("Receive one double data");
            System.out.println("d="+d);
        }
        void receive(String s){
            System.out.println("Receive a string");
            System.out.println("s="+s);
        }
    }
    public class MethodOverloadingTest{
        public static void main(String args[]){
            MethodOverloading mo=new MethodOverloading()；
            mo.receive(1);
            mo.receive(2,3)；
            mo.receive(12.56)；
            mo.receive("very interesting,isn't it?");
        }
    }
```



运行结果是：

```
C:\>java MethodOverloadingTest
Receive one int data
i=1
Receive two int datum
x=2   y=3
Receive one double data
d=12.56
Receive a string
s= very interesting,isn't it?
```

编译器根据参数的个数和类型来决定当前所使用的方法。注意，如果两个方法的声明中，参数的类型和个数均相同，只是返回的类型不同，则编译时会产生错误，即返回类型不能用来区分重载的方法。



从【例 5-5】可以看出，重载虽然表面上没有减少编写程序的工作量，但实际上重载便利程序的实现方式变得很简单，只需要记住一个方法名，它就可以根据不同的输入类型选择该方法的不同的版本。重载和调用如图 5-2 所示。

重载	调用
void receive(int i){……}	<-----receive(1)
void receive(int x,int y){……}	<-----receive(2,3)
void receive(double d) { ……}	<-----receive(12.56)
void receive(String s) { ……}	<-----receive("very interesting,isn't it?")

图 5-2　重载与调用关系

5.1.6　构造方法



在 Java 语言中，当一个对象被创建时，它的成员可以由一个构造方法(函数)初始化。被自动调用的专门的初始化方法称为构造方法，它是一种特殊的方法，为了与其他的方法区别，构造方法的名字与类的名字相同，而且不返回任何数据类型。和其他的方法一样，它在类中定义。一般将构造方法声明为公共的 public 型，如果声明为 private 型，那么就不能够创建对象的实例了，因为构造方法是在对象的外部被默认地调用。构造方法对对象的创建是必须的。实际上，Java 语言为每一个类提供了一个默认的构造方法，也就是说，每个类都有构造方法，用来初始化该类的一个新的对象。如果不定义一个构造方法，Java 语言将调用它提供的默认的构造方法对一个新的对象进行初始化。在构造方法的实现中，也可以进行方法重载。

【例 5-6】构造方法的实现。

```
class Point{
        int x,y;
        point( ){//定义构造方法
            x=0;
            y=0;
        }
        point(int x, int y){ //构造方法的重载
            this.x=x;
            this.y=y;
        }
    }
```

上例中，对类 point 实现了两个构造方法，方法名均为 point，与类名相同。而且使用了方法重载，根据不同的参数分别对点的 x、y 坐标赋与不同的初值。



在【例 5-4】中，曾用 init( )方法对点 x、y 坐标进行初始化。两者完成相同的功能，那么用构造方法的好处在哪里呢？当用运算符 new 为一个对象分配内存时，要调用对象的构造方法，而当构建一对象时，必须用 new 为它分配内存。因此，用构造方法进行初始化避免了在生成对象后每次都要调用对象的初始化方法。如果没有实现类的构造方法，则 Java 运行时系统会自动提供默认的构造方法，它没有任何参数。另外，构造方法只能由 new 运算符调用。

5.1.7　main( )方法

main()方法是 Java 应用程序(Application)必须具备的方法。格式是：

```
public static void main(String args[]){
    ……
}
```

所有独立的 Java 应用程序都从 main()开始执行。把 static 放在方法名前就使方法变为静态的方法，即类方法而非实例方法。



5.1.8　finalize( )方法

在对对象进行垃圾收集前，Java 运行时系统会自动调用对象的 finalize()方法来释放系统资源，如打开的文件或 socket。该方法的声明必须如下所示：

```
protected void finalize( ) throws throwable
```

finalize( )方法在类 java.lang.Obect 中实现，它可以被所有类使用。如果要在一个所定义的类中实现该方法以释放该类所占用的资源(即要重载父类的 finalize( )方法)，则在对该类所使用的资源进行翻译后，一般要调用父类的 finalize( )方法以清除对象使用的所有资源，包括由于继承关系而获得的资源。通常的格式应为：

```
protected void finalize( ) throws throwable{
        ...                //clean Up code for this class
}
```

【例 5-7】finalize 方法举例。

```
class Myclass{
  int m_DataMember1;
  float m_DataMember2;
  public myClass(){
    m_DataMember1=1;   //初始化变量
```

```
    m_DataMember2=7.25;
  }
  void finalize(){        //定义 finalize 方法
    m_DataMember1=null; //释放内存
    m_DataMember2=null;
  }
}
```

注意如果不定义 finalize 方法，Java 将调用它提供的默认的 finalize 方法进行扫尾工作。

5.1.9 包



利用面向对象技术开发一个实际的系统时，通常需要定义许多类一起协同工作。为了更好地管理这些类，Java 中引入了包的概念。包是类和接口定义的集合，就像文件夹或目录把各种文件组织在一起，使硬盘保存的内容更清晰、更有条理一样。Java 中的包把各种类组织在一起，使得程序功能清楚、结构分明。更重要的是包可用于实现不同程序间类的重用。

包是一种松散的类和接口的集合。一般不要求处于同一个包中的类或者接口之间有明确的联系，如包含、继承等关系，但是由于同一包中的类在默认情况下可以互相访问，所以为了方便编程和管理，通常把需要在一起协同工作的类和接口放在一个包里。Java 平台将它的各种类汇集到功能包中。用户可以使用由系统提供的类库，也可以编写自己的类。

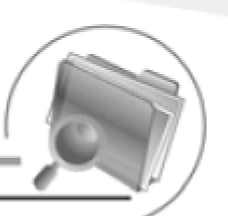
在 Java 语言中包含的标准包如表 5-1 所示。

表 5-1　标准的 Java 包列表

包	功 能 描 述
java.applet	包含一些用于创建 Java 小应用程序的类
java.awt	包含一些编写平台无关的图形用户界面(GUI)应用程序的类。它包含几个子包，包括 java.awt.peer 和 java.awt.image
java.io	包含一些用作输入输出(I/O)处理的类。数据流就包含在这里
java.1ang	包含一些基本 Java 类。java.1ang 是被隐式地引入的，所以用户不必引入它的类
java.net	包含用于建立网络连接的类。与 java.io 同时使用以完成与网络有关的读和写
java.util	包含一些其他的工具和数据结构，如编码、解码、向量和堆栈等

Java 编译器为每个类生成一个字节码文件，且文件名与类名相同，因此同名的类有可能发生冲突。为了解决这一问题，Java 提供包来管理类名空间。包实际上提供了一种命名机制和可见性限制机制。

Java 中的包是一组类，要想使某个类成为包的成员，必须使用 package 语句声明它。而且它应该是整个.java 文件的第一条语句，指明该文件中定义的类所在的包。若缺省该语句，则指定为无名包。其格式为：



```
package 包名;
```

Java 编译器把包对应于文件系统的目录管理。例如：名为 myPackage 的包中，所有类文件都存储在目录 myPackage 下。同时，package 语句中，用.来指明目录的层次，例如：

```
package java.awt.image;
```

指定这个包中的文件存储在目录 path/java/awt/image 下。

包层次的根目录 path 是由环境变量 CLASSPATH 来确定的。

为了能使用 Java 中已提供的类，需要用 import 语句来引入所需要的类。import 语句的格式为：

```
import packagel[.package2...].(classname|*);
```

其中 packagel[.package2…]表明包的层次，与 package 语句相同，它对应于文件目录，classname 则指明所要引入的类。

Java 编译器为所有程序自动引入包 java.lang，因此不必用 import 语句引入它包含的所有的类，但是若需要使用其他包中的类，必须用 import 语句引入。

如果要从一个包中引入多个类，则可以用星号(*)代替。例如：

```
import java.awt.*;
```

如果只需要某个包中的一个类，这时可以只装入这个类，而不需要装载整个包。装载一个类可以使用语句：

```
import java.util.Date;
```

5.2 对象

Java 程序定义类的最终目的是使用它，像使用系统类一样，程序也可以继承用户自定义类或创建并使用自定义类的对象。把类实例化，可以生成多个对象，这些对象通过消息传递来进行交互(消息传递即激活指定的某个对象的方法以改变其状态或让它产生一定的行为)，最终完成复杂的任务。

一个对象的生命期包括 3 个阶段：创建、使用和清除。

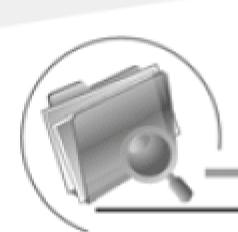
5.2.1 对象的创建

对象的创建包括声明、实例化和初始化 3 方面的内容。通常的格式为：

```
type ObectName=new type([paramlist]);
```

(1) type objectName 声明了一个类型为 type 的对象(objectName 是一个引用，标识该 type 类型的对象)，其中 type 是引用类型(包括类和接口)，对象的声明并不为对象分配内存空间。(但





objectName 分配了一个引用的空间。)

(2) 运算符 new 为对象分配内存空间，实例化一个对象。new 调用对象的构造方法，返回对该对象的一个引用(即该对象所在的内存地址)。用 new 可以为一个类实例化多个不同的对象。这些对象分别占用不同的内存空间，因此改变其中一个对象的状态不会影响其他对象的状态。

(3) 生成对象的最后一步是执行构造方法，进行初始化。由于对构造方法可以进行重写，所以通过给出不同个数或类型的参数会分别调用不同的构造方法。如果类中没有定义构造方法，系统会调用默认的空构造函数。

【例 5-8】定义类并创建类的对象。

```
class Computer{
      String Owner;   //成员变量
      public static void main(String args[]){
      }
      void set_Owner(String owner){     //成员方法
   Owner=owner;
      }
      void show_Owner( ){
    System.out.println("这台电脑是:"+Owner+"的");
      }
}

class DemoComputer{
      public static void main(String args[]){
        System.out.println("使用类");
    Computer MyComputer=new Computer( );    //生成 Computer 类的对象 MyComputer
    Mycomputer.set_Owner("知识工程教研室");
    Mycomputer.show_Owner( );
      }
}
```



这里定义了 Computer 和 DemoComputer 两个类，其中 Computer 和 DemoComputer 是类的名称，用户可以自己命名，但要注意不能和保留字冲突。定义好之后，Computer、DemoComputer 就可以看成一个数据类型来使用，这种数据类型的变量就是对象，例如下面的定义：

```
Computer MyComputer=new Computer( );
```

等价于：

```
Computer MyComputer;
MyComputer=new Computer( );
```

其中 MyComputer 是对象的名称，它是一个属于 Computer 类的对象，所以能够调用 Computer 类中的 set_Owner( )、show_Owner( )方法。



虽然 new 运算符返回对一个对象的引用，但与 C / C++中的指针不同，对象的引用是指向一个中间的数据结构，它存储有关数据类型的信息以及当前对象所在的堆的地址，而对于对象所在的实际的内存地址是不可操作的，这就保证了安全性。

5.2.2 对象的使用

对象的使用包括引用对象的成员变量和方法，通过运算符. 可以实现对变量的访问和方法的调用。

先定义一个类 Point，它在【例 5-6】的定义中添加一些内容。

【例 5-9】对象的使用示例。

```
class Point{
  int x,y;
  String name="a point";
  Point( ){
    x=0;
    y=0;
  }
  Point(int x,int y,String name){
    this.x=x;
    this.y=y;
    this.name=name;
  }
  int getX(){
    return x;
  }
 int getY(){
  return y;
 }
 void move(int newX,int newY){
  x=newX;
  y=newY;
 }
 Point newPoint(String name){
  Point newP=new Point(-x,-y,name);
  return newP;
 }
 boolean equal(int x,int y){
  if(this.x==x&&this.y==y)
    return true;
  else
```



```
        return false;
    }
    void print( ){
      System.out.println(name+":   x="+x+"   y="+y);
    }
}
public class UsingObject{
       public static void main(String args[]){
      Point p=new Point();
      p.print( );
      p.move(50,50);
      System.out.println("****after moving****");
      System.out.println("Get x and y directly");
      System.out.println("x="+p.x+"   y="+p.y);
      System.out.println("or Get x and y by calling method");
      System.out.println("x="+p.getX( )+"   y="+p.getY( ));
      if(p.equal(50,50))
        System.out.println("I like this point!");
      else
        System.out.println("I hate it!");
      p.newPoint("a new point").print( );
      new Point(10,15,"another new point").print( );
        }
}
```

运行结果为：

```
C:\>java UsingObject
a point:   x=0   y=0
****after moving****
Get x and y directly
x=50   y=50
or Get x and y by calling method
x=50   y=50
I like this point!
a new point:   x=-50   y=-50
another new point:   x=10   y=15
```

(1) 引用对象的变量

要访问对象的某个变量，其格式为：

```
objectReference.variable
```



其中 objectReference 是对象的一个引用，它可以是一个已生成的对象，也可以是能够生成对象引用的表达式。

例如，用 Point p=new Point( ); 生成了类 Point 的对象 p 后，可以用 p.x，p.y 来访问该点的 x，y 坐标，如：

```
p.x = 10;
p.y = 20;
```

或者用 new 生成对象的引用，然后直接访问，如：

```
tx = new point( ).x;
```

(2) 调用对象的方法

要调用对象的某个方法，其格式为：

```
objectReference.methodName ([paramlist]);
```

例如要移动类 Point 的对象 p，可以用：

```
p.move(30,20);
```

或者用 new 生成对象的引用，然后直接调用它的方法，如：

```
new point( ).move (30,20);
```



5.2.3　对象的清除

对象的清除，即系统内无用单元的收集。在 Java 管理系统中，使用 new 运算符来为对象或变量分配存储空间。程序设计者不用刻意在使用完对象或变量后，删除该对象或变量来收回它所占用的存储空间。Java 运行时系统通过垃圾收集周期性地释放无用对象所使用的内存，完成对象的清除。当不存在对一个对象的引用(当前的代码段不属于对象的作用域或把对象的引用赋值为 null，如 p=null)时，该对象成为一个无用对象。Java 运行系统的垃圾收集器自动扫描对象的动态内存区，对被引用的对象加标记，然后把没有引用的对象作为垃圾收集起来并释放它，释放内存是系统自动处理的。该收集器使得系统内存的管理变得简单、安全。垃圾收集器作为一个线程运行。当系统的内存用尽或程序中调用 System.gc() 要求进行垃圾收集时，垃圾收集线程与系统同步运行。否则垃圾收集器在系统空闲时异步地执行。在 C 中，通过 free 来释放内存，C++中则通过 delete 来释放内存，这种内存管理方法需要跟踪内存的使用情况，不仅复杂而且还容易造成系统的崩溃，Java 采用自动垃圾收集进行内存管理，使程序员不需要跟踪每个生成的对象，避免了上述问题的产生，这是 Java 的一大优点。

当下述条件满足时，Java 内存管理系统将自动完成收集内存工作。

(1) 当堆栈中的存储器数量少于某个特定水平时。

(2) 当程序强制调用系统类的方法时。

(3) 当系统空闲时。

当条件满足时，Java 运行环境停止程序操作，恢复所有可能恢复的存储器。在一个对象作为垃圾(不被引用)被收集前，Java 运行时系统会自动调用对象的 finalize( )方法，使它清除自己所使用的资源。

5.3 访问控制符

访问控制符是一组限定类、属性和方法是否可以被程序里的其他部分访问和调用的修饰符。具体地说，类及其属性和方法的访问控制符规定了程序其他部分能否访问和调用它们。这里的其他部分是指程序中该类之外的其他类或函数。

无论修饰符如何定义，一个类总能够访问和调用它自己的成员，但是这个类之外的其他部分能否访问变量或方法，就要看该变量和方法以及它所属的类的访问控制符了。

类的访问控制符只有一个 public。成员变量和成员方法的访问控制符有 3 个，分别为 public、protected、private。另外还有一种没有定义专门的访问控制符的默认情况。

5.3.1 类的访问控制符

(1) 公共访问控制符 public

Java 中类的访问控制符只有一个：public，即公共类。一个类被声明为公共类，声明它可以被所有其他类所访问和引用，这里的访问和引用是指这个类作为整体是可见和可使用的。程序的其他部分可以创建这个类的对象，访问这个类可用的成员变量和方法。Java 的类可以通过包的概念来组织，处于同一个包中的类可以不需任何说明而方便地互相访问和引用，而对于在不同包中的类，一般说来，它们相互之间是不可见的，当然也不可能互相引用。但是，当一个类被声明为 public 时，它就具有了被其他包中的类访问的可能性，只要在程序中使用 import 语句引入 public 类，就可以访问和引用这个类。

一个类作为整体对于程序的其他部分可见，并不代表类的所有成员变量和方法也同时对于程序的其他部分可见。类的成员变量和方法能否为所有其他类所访问，还要看这些域和方法自己的访问控制符。类中被设定为 public 的方法是这个类对外的接口部分，程序的其他部分通过调用它们达到与当前类交换信息，传递消息甚至影响当前类的作用，从而避免了程序的其他部分直接去操作类的数据。

如果一个类中定义了常用的操作，希望能作为公共工具供其他的类和程序使用，则也应该把类本身和这些方法都定义成 public，如 Java 类库中的那些公共类和它们的公共方法。另外，每个 Java 程序的主类都必须是 public 类，也是基于相同的原因。

(2) 缺省访问控制符

若一个类没有访问控制符，说明它具有缺省的访问控制特性。该访问控制规定这样的类只



能被同一个包中的类访问和引用，而不能被其他包中的类使用，这种访问特性又称包访问性。通过声明类的访问控制符可以使整个程序结构清晰、严谨，减少可能产生的类之间的干扰和错误。

5.3.2　对类成员的访问控制

类中成员变量和成员方法的声明中，有 public、protected、private 这些修饰词，这些修饰词的作用是对类的成员施以一定的访问权限限定，实现类中成员在一定范围内的信息隐藏。Java 语言中，提供 4 种不同的访问权限，以实现 4 种不同范围的访问能力。表 5-2 中说明了这些限定词的作用。

表 5-2　Java 中类的限定词的作用范围比较

	同一个类中	同一个包中	不同包中的子类	不同包中的非子类
private	★			
default	★	★		
protected	★	★	★	
public	★	★	★	★



从表 5-2 中可以看出，类总是可以访问该类自己的成员。下面详细说明表 5-2 中的各种访问等级。

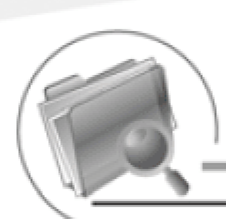
(1) private

限制性最强的访问等级是 private。类中限定为 private 的成员只能被这个类本身访问。如果 private 的方法被外部类所调用，就会便利运行的程序或者对象的状态处于非正常状态。下面的类包含了一个 private 成员变量和一个 private 方法。

```
class Alpha{
private int iamprivate;              //private 成员变量
private void privateMethod(){        //private 成员方法
      System.out.println("privateMethod");
  }
 }
```

在 Alpha 类中，其对象或方法可以检查或者修改 iamprivate 变量，也可以调用 privateMethod 方法，但在 Alpha 类外的任何地方都不行。比如，以下的 Beta 类中就不能通过 Alpha 对象访问它的私有变量或者方法：

```
class Beta{
void accessMethod(){
    Alpha a=new Alpha();
    a.iamprivate=10;      //非法
    a.privateMethod();   //非法
```




```
    }
}
```

当试图访问一个没有权限访问的成员变量的时候，编译器就会给出错误信息并拒绝继续对源程序进行编译。同样地，如果试图访问一个不能访问的方法，也将导致编译错误。

一个类不能访问其他类对象的 private 成员，但是同一个类的两个对象能否互相访问 private 成员呢？下面举例来进行解释。

```
class Alpha{
private int iamprivate;
boolean isEqualTo(Alpha anotherAlpha){
  if(this.iamprivate==anotherAlpha.iamprivate)
    return true;
  else
    return false;
 }
}
```



同一个类的不同对象可以访问对方的 private 成员变量或调用对方的 private 方法，这是因为访问保护是控制在类的级别上，而不是在对象的级别上。另外，对于构造方法，我们也可以限定它为 private。如果一个类的构造方法声明为 private，则其他类不能生成该类的一个实例。

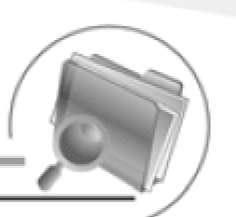
(2) default

类中不加任何防问权限限定的成员属于默认的(default)访问状态，可以被这个类本身和同一个包中的类所访问。这个访问级别是假设在相同包中的类是互相信任的。例如：

```
package Greek;
public class Alpha{
  int iamprivate;
  void packageMethod( ){
        System.out.println("packageMethod");
    }
}
```

Alpha 类可以访问自己的成员，同时所有定义在与 Alpha 同一个包中的类也可以访问这些成员。如 Alpha 和 Beta 都定义为 Greek 包的一部分，则 Beta 可以合法访问 Alpha 的成员。

```
package Greek;
class Beta{
  void accessMethod( ){
        Alpha a=new Alpha( );
        a.iamprivate=10;          //合法
        a.protectedMethod( );     //合法
  }
}
```



(3) protected

类中限定为 protected 的成员可以被这个类本身、它的子类以及同一个包中所有其他的类访问。因此，在允许类的子类和相关的类访问而杜绝其他不相关的类访问时，可以使用 protected 访问级别，并且把相关的类放在一个包中。

```
package Greek;
public class Alpha{
  protected int iamprivate;
  protected void privateMethod( ){
        System.out.println("protectedMethod");
    }
}
```

再假设有个类 Gamma 也声明为包 Greek 的一个成员。Gamma 类可以合法访问 Alpha 对象的成员变量 iamprivate 并且可以合法调用它的 protectedMethod。

```
package Greek;
class Gamma{
  void accessMethod( ){
        Alpha a=new Alpha( );
        a.iamprivate=10;          //合法
        a.protectedMethod( );     //合法
  }
}
```

下面再来研究一下 protected 是怎样影响 Alpha 的子类的访问。

首先引入一个新的类 Delta，它继承类 Alpha，但是处在另一个包 Latin 中。这个 Delta 类不仅可以访问 Delta 类的成员 iamprivate 和 protectedMethod，而且可以访问它的父类。但 Delta 类不能访问 Alpha 类的对象中的 iamprivate 或者 protectedMethod。

```
package Latin;
import Greek.*;
class Delta extends Alpha{
  void accessMethod(Alpha a,Delta d){
        a.iamprivate=10;          //非法
        d.iamprivate=10;          //合法
        a.protectedMethod();      //非法
        d.protectedMethod();      //合法
  }
}
```

处在不同包中的子类虽然可以访问父类中限定为 protected 的成员，但这时访问这些成员的



对象必须具有子类的类型或者是子类的子类类型，而不能是父类类型。

(4) public

在 Java 中，类中限定为 public 的成员可以被所有的类访问。一般情况下，一个成员只有在外部对象使用后不会产生不良的后果的时候，才声明为公共的。为了声明一个公共的成员，要使用关键字 public，例如：

```
package Greek;
public class Alpha{
  public int iampublic;
  public void publicMethod( ){
        System.out.println("publicMethod");
    }
}
```

现在重新编写 Beta 类再将它放置到不同的包中，并且要确保它跟 Alpha 毫无关系。

```
package Roman;
import Greek.*;
class Beta{
  void accessMethod( ){
        Alpha a=new Alpha( );
        a.iampublic=10;          //合法
        a.publicMethod( );       //合法
  }
}
```

从上面的代码段可以看出，Beta 可以合法地访问和修改在 Alpha 类中的 iampublic 变量，并且可以合法地调用方法 publicMethod。

(5) 访问控制符小结

访问控制符是一组限定类、变量或方法是否可以被其他类访问的修饰符。

① 公共访问控制符(public)：

public 类，公共类，可以被其他包中类引入后访问。

public 方法，是类的接口，用于定义类中对外可用的功能方法。

public 变量，可以被其他类访问。

② 缺省访问控制符的类、变量、方法：具有包访问性(只能被同一个包中的类访问)。

③ 私有访问控制符(private)：修饰变量或方法，只能被该类自身所访问。

④ 保护访问控制符(protected)：修饰变量或方法，可以被类自身、同一包中的类、任意包中该类的子类所访问。



5.4　继承与多态

继承是面向对象程序设计设计中的一种重要手段，通过继承可以更有效地组织程序结构，明确类间关系，并充分利用已有的类来创建新类，以完成更复杂的设计、开发。多态则可以统一多个相关类对外的接口，并在运行时根据不同的情况执行不同的操作，提高类的抽象度和灵活性。

5.4.1　子类、父类与继承机制

(1) 继承的概念

在面向对象技术中，继承是最具有特色的一个特点。继承表示存在于面向对象程序中的两个类之间的一种关系。通过继承可以实现代码的复用，使程序的复杂性线性地增长，而不是随规模增大呈几何级数增长。当一个类自动拥有另一个类的所有属性(变量和方法)时，就称这两个类之间具有继承关系。被继承的类称为父类，继承了父类的所有属性的类称为子类。如图 5-3 所示，圆类可继承点类的所有属性，并以继承的坐标点为圆心，自定义的成员变量为半径完成各种操作。



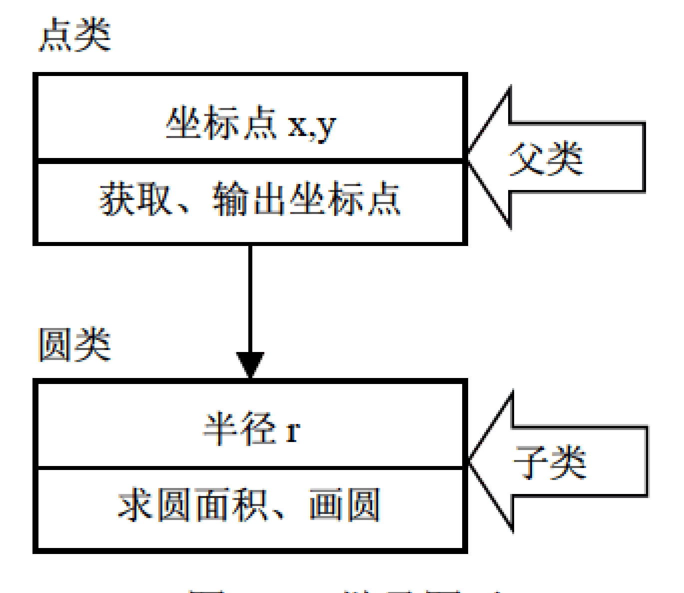


图 5-3　继承图示

继承是一种由已有的类创建新类的机制。父类和子类间的关系具有：共享性、层次性、差异性。由于父类代表了所有子类的共性，而子类既可继承其父类的共性，又可以具有本身独特的个性，在定义子类时，只要定义它本身所特有的属性与方法就可以了。从这个意义上看，继承可理解为：子类的对象可拥有父类的全部属性和方法，但父类的对象却不能拥有子类的对象的全部属性和方法。

Java 语言出于安全、可靠性的考虑，仅提供了单继承机制。Java 程序中的每个类只有一个直接的父类，而 Java 多继承的功能则是通过接口方式来间接实现的。

(2) 类的层次

Java 语言的类具有层次的结构，实质上 Java 的系统定义类就是一个类层次，如图 5-4 所示。Object 类定义和实现了 Java 系统所需要的众多类的共同行为，它是所有类的父类。Object 是一



个根类，所有的类都是由这个类继承、扩充而来的，这个 Object 类定义在 java.lang 包中。

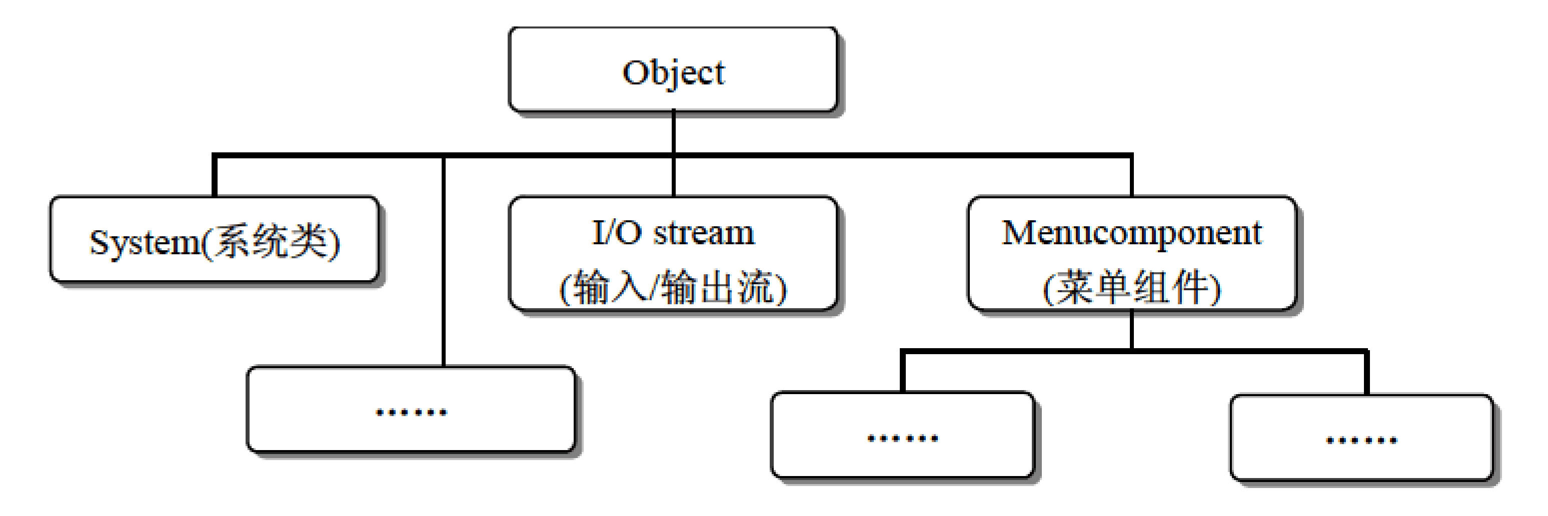


图 5-4　Java 语言中类的层次

从图 5-4 中可以看出，位于最高层次的是 Object 类，也称为对象基类或超类或父类。在 Object 类的下层有许多子类，也称为导出类或派生类。事实上，每个子类又可以有它下面的许多子类，从而形成一个规模很大的类层次结构。



Java 中，所有的类都是通过直接或间接地继承 java.lang.Object 得到的。继承而得到的类为子类，被继承的类为父类，父类包括所有直接或间接被继承的类。子类继承父类的状态和行为，同时也可以修改父类的状态或重写父类的行为，并添加新的状态和行为，Java 中不支持多重继承。

(3) 创建子类

通过在类的声明中加入 extends 子句来创建一个类的子类，其格式如下：

```
class 子类名 extends 父类名称{
    ……
}
```

如果父类又是某个类的子类，则所创建的子类同时也是该类的(间接)子类。子类可以继承所有父类的内容。如果缺省 extends 子句，则该类为 java.lang.Object 的子类。子类可以继承父类中访问权限设定为 public、protected、default 的成员变量和方法。但是不能继承访问权限为 private 的成员变量和方法。

【例 5-10】由圆类 circle 继承点类 point。即先定义一个点类，然后由点类派生一个圆类。

```
  class  Point{
      int x,y;
      void getxy(int i, int j){
          x=i;
          y=j;
          }
  }
class Circle extends Point{
      double r;
      double area( ){
```



```
            return 3.14*r*r;
        }
    }
```

在定义子类时用 extends 关键字指明新定义类的父类，就在两个类之间建立了继承关系。新定义的类称为子类，它可以从父类那里继承所有非 private 的属性和方法作为自己的属性和方法。

【例 5-11】应用继承性的实例。

```
class Student{          //自定义“学生”类
  int stu_id;           //定义属性：学生学号
  void set_id(int id){  //定义方法：设置学号
    stu_id=id;
  }
  void show_id(){       //定义方法：显示学号
    System.out.println("the student ID is:"+stu_id);
 }
}
class UniversityStudent extends Student{  //定义 UniversityStudent 是 Student 的子类
  int dep_number;      //定义子类特有的属性变量：系别号
  void set_dep(int dep_num){ //定义子类特有的方法
    dep_number=dep_num;
  }
  void show_dep( ){
    System.out.println("the dep_number is:"+dep_number);
  }
  public static void main(String args[]){
    UniversityStudent Lee=new UniversityStudent();
    Lee.set_id(2007070130);  //继承父类学生的属性
    Lee.set_dep(701);     //使用本类的属性
    Lee.show_id();       //继承父类学生的方法
    Lee.show_dep();     //使用本类的方法
}
}
```

学生有小学生、中学生和大学生之分，因此，学生可以作为具有共性的父类，而大学生则是学生的一种，具有特殊性，因此可以作为子类。这样，大学生子类应继承学生的所有属性和方法，而本身还可以有自身的特殊的属性和方法。

(4) 成员变量的隐藏和方法的覆盖

下面先看一个例子。



【例 5-12】成员变量的隐藏和方法的覆盖示例。

```
class SuperClass{
int x;
    ……
    void setX(){
        x=0;
    }
    ……
}
class SubClass extends SuperClass{
    int x;          //hide x SuperClass
    ……
    void setX(){    //override method setX() in SuperClass
        x=5;
    }
    ……
}
```

该例中，SubClass 是 SuperClass 的一个子类。其中声明了一个和父类 SuperClass 同名的变量 x，并定义了与之相同的方法 setX，这时在子类 SubClass 中，父类的成员变量 x 被隐藏，父类的方法 setX 被重写。于是子类对象所使用的变量 x 为子类中定义的 x，子类对象调用的方法 setX()为子类中所实现的方法。子类通过成员变量的隐藏和方法的重写可以把父类的状态和行为改变为自身的状态和行为。

子类重新定义一个与父类那里继承来的成员变量完全相同的变量，称为成员变量的隐藏。方法的覆盖是指子类重定义从父类继承来的一个同名方法，此时子类将清除父类方法的影响。

注意，子类在重新定义父类已有的方法时，应保持与父类完全相同的方法头声明，即应与父类有完全相同的方法名、相同的参数表和相同的返回类型。

(5) super

子类在隐藏了父类的成员变量或重写了父类的方法后，常常还要用到父类的成员变量，或在重写的方法中使用父类中被重写的方法以简化代码的编写，这时就要访问父类的成员变量或调用父类的方法，Java 中通过 super 来实现对父类成员的访问。Java 中，this 用来引用当前对象，与 this 类似，super 用来引用当前对象的父类。

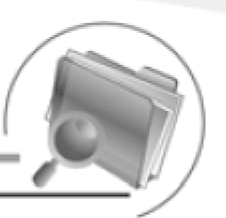
super 的使用有 3 种情况：

① 用来访问父类被隐藏的成员变量，如：

```
super.variable
```

② 用来调用父类中被重写的方法，如：

```
super.Method([paramlist]);
```



③ 用来调用父类的构造方法，如：

```
super(rparamlist));
```

下面通过一个例子来说明 super 的使用，以及成员变量的隐藏和方法的重写。

【例 5-13】super 的使用示例。

```
class SuperClass{
      int x;
      superClass( ) {
        x=3;
        System.out.println("in superClass: x = "+x);
      }
      void doSomething( ){
        System.out.println("in superClass.doSomething( )");
      }
    }
    class subclass extends superClass {
      int x;
      subclass( ) {
        super( );            //call constructor of superClass
        x=5;
        System.out.println("in subclass : x = "+x);
      }
      void doSomething( ){
        super.doSomething( ); //call method of superClass
        System.out.println("in subClass.doSomething( )");
        System.out.println("super.x = "+super.x+" sub.x = "+x);
      }
    }
    public class Inheritance {
      public static void main( String args[ ] ){
        subclass subC = new subclass( );
        subC.doSomething( );
      }
    }
```

运行结果为：

```
C:\>java Inheritance
in superClass: x = 3
in subclass : x = 5
```



计算机基础与实训教材系列

```
in superClass.doSomething( )
in subClass.doSomething( )
super.x = 3 sub.x = 5
```

通常，在实现子类的构造方法时，先调用父类的构造方法。在实现子类的 finalize( )方法时，最后调用父类的 finalize( )方法，这符合层次化的观点以及构造方法和 finalize( )方法的特点。即初始化过程总是由高级向低级，而资源过程应从低级向高级进行。

(6) 继承性设计原则

在面向对象继承性设计中，有以下几条重要且有用的原则：

① 尽量将公共的操作和属性放在父类中。这是通过类的继承实现代码重用的基本要求，通过定义父类中的方法，使得所有的子类都能重用这些代码，对于提高程序开发效率是有很大好处的。

② 利用继承实现问题模型中的“子类是父类中的一种”的关系。

③ 子类继承父类的前提是父类中的方法对子类都是可用的。如果要说明一个类继承另一个类，就必须考虑父类的方法是否对子类都是适用的，如果不适用的方法很多，继承就失去了意义。

5.4.2 多态性

1. 多态性的概念

多态性是由封装性和继承性引出的面向对象程序设计语言的另一特征。在面向过程的程序设计中，各函数是不能重名的，否则在用名字调用时，就会产生歧义和错误。而在面向对象的程序设计中，有时却需要利用这样的“重名”现象来提高程序的抽象度和简洁性。

多态性是指同名的不同方法在程序中共存。即为同一个方法定义几个版本，运行时根据不同情况执行不同的版本。调用者只需使用同一个方法名，系统会根据不同情况，调用相应的不同方法，从而实现不同的功能。多态性又被称为“一个名字，多个方法”。

在 Java 语言中，多态性的实现有两种方式：

(1) 覆盖实现多态性

通过子类对继承父类方法的重定义来实现。使用时注意：在子类重定义父类方法时，要求与父类原型(参数个数、类型、顺序)完全相同。

在覆盖实现多态性的方式中，如何区别这些同名的不同方法呢？由于这些方法是存在于一个类层次结构的不同子类中的，在调用方法时只需要指明调用哪个类(或对象)的方法，就很容易把它们区分开来。

(2) 重载实现多态性

通过定义类中的多个同名的不同方法来实现。编译时是根据参数(个数、类型、顺序)的不同来区分不同方法的。



由于重载发生在同一个类中，不能再用类名来区分不同的方法了，所以在重载中采用的区分方法是使用不同的形式参数表，包括形式参数的个数不同、类型不同或顺序的不同。本书在前面讲过方法重载的概念，即完成一组相似功能的方法可具有相同的方法名，只是方法接收的参数不同。在前面举的许多例子中都用到了打印输出方法 println( )，就是一个典型的重载的方法，我们可以提供给该方法不同的参数：int、double、String 等类型，程序会根据参数的不同来调用相应的方法、打印不同类型的数据。具体调用哪个被重载的方法，是由编译器在编译阶段静态确定的，所以说重载实现多态性体现了静态的多态性。

2. 覆盖实现多态性

子类对象可以作为父类对象使用，这是由于子类通过继承具备了父类的所有属性(私有的除外)。所以，在程序中凡是要求使用父类对象的地方，都可以用子类对象来代替。另外，子类还可以重写父类中已有的成员方法，实现父类中没有的功能。

(1) 重写方法的调用规则

对于重写的方法，Java 运行时系统根据调用该方法的实例的类型来决定选择哪个方法调用。对于类的一个实例，如果子类重写了父类的方法，则运行时系统调用子类的方法。如果子类继承了父类的方法(未重)，则运行时系统调用父类的方法。因此，一个对象可以通过引用子类的实例来调用子类的方法。如【例 5-14】所示。



【例 5-14】重写方法的调用规则示例。

```
class A{
    void callme( ){
        System.out.println("InsideA'scallme( )method");
    }
}

  class B extends A{
    void callme( ){
        System.out.println("Inside B's callme( ) method");
    }
}
public class Dispatch{
    public static void main(String args[]){
        A a=new B( ); a.callme( );
    }
}
```

运行结果为：

```
 C:\>java Dispatch
Inside B's callme( ) method
```

在【例 5-14】中，声明了 A 类型的变量 a，然后用 new 建立 A 类型的子类 B 的一个实例 b，并把对该实例的一个引用存储到 a 中，Java 运行时系统分析该引用是类型 B 的一个实例，因此调用子类 B 的 callme 方法。

用这种方式可以实现运行时的多态，它体现了面向对象程序设计中的代码复用和鲁棒性。已经编译好的类库可以调用新定义的子类的方法而不必重新编译，而且还提供了一个简明的抽象接口，如【例 5-14】中，如果增加几个 A 的子类的定义，则用 a．callme( )可以分别调用多个子类的不同的 callme( )方法，且只需分别用 new 生成不同子类的实例即可。

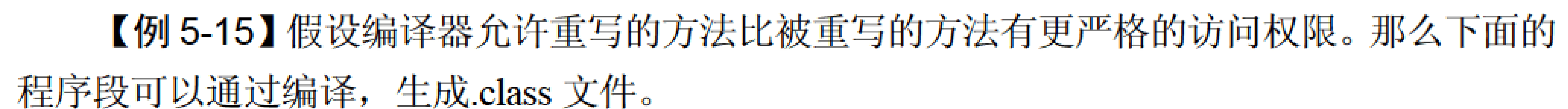
(2) 方法重写时应遵循的原则

方法重写的两个原则：

- 改写后的方法不能比被重写的方法有更严格的访问权限。
- 改写后的方法不能比被重写的方法产生更多的例外。

进行方法重写时必须遵从这两个原则，否则编译器会指出程序出错。编译器加上这两个限定，是为了与 Java 语言的多态性特点一致而作出的。可以通过对下面的程序段的分析得出这些结论。





【例 5-15】假设编译器允许重写的方法比被重写的方法有更严格的访问权限。那么下面的程序段可以通过编译，生成.class 文件。

```
class Parent{
   public void fimction( ){
   }
}
class Child extends Parent{
   private void function( ){
   }
}
public class OverriddenTest {
   public static void main(String args[]){
      Parent p1 = new Parent( );
      Parent p2 = new Child( );
      p1.function( );
      p2.function( );
   }
}
```

当程序执行到 p2.function( )时，由于 p2 指向的是 Child 类的对象，p2.function( )会调用 Child 类的 function( )方法，由于该类的 funtion( )方法的访问权限为 private 的，所以会导致访问权限冲突。产生这种错误的原因在于子类中重写的方法 function( )比父类中被重写的方法有更严格的访问权限。为了避免这种错误的产生，Java 语言规定不允许这样使用方法重写，否则会在编译时产生错误。



第 2 点规则也是与对象的多态性有关的，这样限定是出于对程序的健壮性的考虑，为了避免程序中有应该捕获而未捕获的例外。涉及例外处理的部分，将在后面介绍。

3. 重载实现多态性

重载实现多态性是通过定义类中的多个同名的不同方法来实现的。编译时则根据参数(个数、类型、顺序)的不同来区分不同的方法。通过重载可定义多种同类的操作方法，调用时根据不同需要选择不同的操作。

下面的这个程序创建了一个重载的方法。它用一个简单的类定义开始，定义了一个叫做 MyRect 的类，该类中定义了矩形，用 4 个实例变量来定义这个矩形的左上角和右下角的坐标，x1、y1、x2、y2。另外定义了 3 个同名的不同 buildRect()方法为这些实例变量设置值。

【例 5-16】重载实现多态性举例。

```
import java.awt.Point;
class MyRect{
 int x1=0;
 int y1=0;
 int x2=0;
 int y2=0;
 }
 MyRect buildRect(int x1,int y1,int x2,int y2){
  this.x1=x1;
  this.y1=y1;
  this.x2=x2;
  this.y2=y2;
  return this;
 }
 MyRect buildRect(Point topLeft,Point bottomRight){
  x1=topLeft.x;
  y1=topLeft.y;
  x2=bottomRight.x;
  y2=bottomRight.y;
  return this;
 }
 MyRect buildRect(Point topLeft,int w,int h){
  x1=topLeft.x;
  y1=topLeft.y;
  x2=(x1+w);
  y2=(y1+h);
  return this;
 }
```



```
  void printRect(){
    System.out.println("MyRect:<"+x1+","+y1);
    System.out.println(","+x2+","+y2+">");
  }
  public static void main(String args[]){
    MyRect rect=new MyRect();
    rect.buildRect(25,25,50,50);
    rect.printRect();
    System.out.println("******");
    rect.buildRect(new Point(10,10),new Point(20,20));
    rect.printRect();
    System.out.println("******");
    rect.buildRect(new Point(10,10),50,50);
    rect.printRect();
    System.out.println("******");
  }
}
```

运行结果是：

```
C: >java MyRect
MyRect:<25,25,50,50>
******
MyRect:<10,10,20,20>
******
MyRect:<10,10,60,60>
******
```

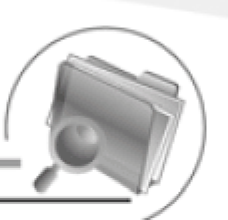
4. 对象状态的确定

既然子类对象可以作为父类对象使用，那么在程序中怎样判断对象究竟属于哪一类呢？在Java 语言中，提供了操作符 instanceof 用来判断对象是否属于某个类的实例。

下面举例说明其用法。下面的程序段中，方法 method( )接收的参数类型为 Employee 型的，Manager 和 Contractor 都是 Employee 的子类，由于子类对象可以作为父类对象使用，所以该方法也可以接收 Manager 和 Contractor 类型的对象，在方法内部，可以通过 instanceof 来判断对象的类型，进而作出不同的处理。

【例 5-17】确定对象状态的应用举例。

```
public void method(Employee e) {
  if(e instanceof Manager){
    ……                    //do something as a Manager
```



```
}
else if(e instanceof Contractor) {
  ……                          //do something as a Contractor
}
  else {
  ……                            //do something else
  }
}
```

5.5　上机练习

本章上机实验主要练习如何使用 Java 进行面向对象编程。其中重点掌握如何定义类的成员变量、成员方法和构造函数、如何定义派生类，如何使用访问修饰符来控制派生类对基类成员变量和方法的访问，如何进行方法重载以及如何定义覆盖方法以实现动态联编。

下面以【例 5-11】为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，将打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择 Java Project 选项，单击 Next 按钮打开 New Java Project 对话框。

(4) 在 New Java Project 对话框的 Project Name 文本框中输入项目名称，如 MyJava，单击 Finish 按钮后 Eclipse 将自动创建一个 Java 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中会出现新建的 MyJava 项目。

(6) 选择【File】|【New】|【Class】命令，弹出 New Java Class 对话框新建一个 Java 类。

(7) 在 New Java Class 对话框的 Package 文本框中输入 Java 类所使用的包名，例如 ch5。

(8) 在 Name 文本框中输入 Java 类的名称。

(9) 选中 public static void main(String[] args)复选框，单击 Finish 按钮完成新类的创建。

(10) Eclipse 会创建一个新的类，包括了基本的类框架。在类的编辑窗口中输入程序代码即可。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，将会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 在 Package Explorer 窗口中右键单击类，从弹出的快捷菜单中选择【Run As】命令，如果 Java 类语法正确，而且具有合法的 main 方法，则在【Run As】命令的子菜单中会出现【Java Application】菜单命令。

(14) 选择【Java Application】命令，并检查控制台输出是否正确。



5.6 习题

5.6.1 填空题

1. 访问控制符包括__________、__________和__________。
2. 类的常量使用__________修饰符定义。
3. 类的函数成员包括__________和__________。
4. 对象使用__________运算符来创建。
5. 方法的定义包括__________和__________两部分。
6. 可以通过定义一个__________类来保护该类不被继承。

5.6.2 选择题

1. 类中的成员默认的访问修饰符是(　　　)。

A. private　　B. protected　　C. public　　D. 空

2. 常量使用(　　　)进行修饰。

A. private　　B. abstract　　C. final　　D. static

3. 如果已经定义了方法 int f(bool b, int i)，则以下方法中，哪一个不是合法的重载方法(　　　)。

A. double f(int i, bool b)　　B. int f(double d, int i)

C. int f(bool b, int i, double d)　　D. double f(bool d, int j)

4. 以下修饰符中，哪一个表示必须由派生类实现(　　　)。

A. private　　B. final　　C. abstract　　D. static

5.6.3 问答题

1. 简述面向对象的几个基本概念：对象、类、继承。
2. 方法覆盖和方法重载有什么区别？



JSP 中的内置对象

学习目标

JSP 内置对象是指不需要声明，就可以直接在 JSP 中使用的对象，它提供了许多进行网络开发所必不可少的功能，甚至可以说，离开这些对象就不能使用 JSP 进行开发工作。

JSP 的内置对象使用户更容易收集通过浏览器请求所发送的信息、响应浏览器以及存储用户信息，本章将详细讲解这些内置对象。

本章重点

- JSP 的内置对象
- out 对象
- request 与 response 对象
- session 对象
- page 对象

6.1 内置对象概述

JSP 的内置对象是在 JSP 运行环境中已经定义好的对象，不需要用户去声明和定义即可直接使用。JSP 的内置对象包括以下几种：out、request、response、session、pageContext、application、config、page 和 exception。这些内置对象都对应 Servlet API 的一些类，实际上对这些对象的使用都将被转化为对相应 Servlet 类的方法的调用。

- Out 对象

Out 对象用于向客户端输出数据。

与 Out 相联系的是 javax.servlet.jsp.jspWriter 类。

⊙ Request 对象

Request 对象用于接收所有从浏览器发送到服务器的请求内的所有信息。

与 Request 相联系的是 javax.servlet.http.HttpServletRequest 类。通过 getParameter 方法可以得到 request 参数，通过 GET、POST、HEAD 等方法可以得到 Request 的类型，通过 cookies、Referer 等可以得到引入的 HTTP 头。

⊙ Response 对象

Response 对象用于向客户端浏览器发送数据，用户可以使用该对象将服务器端的数据发送到客户端浏览器。

与 Response 相联系的是 javax.servlet.http.HttpServletResponse 类。

⊙ Session 对象

Session 对象用于分别保存每一个用户信息的对象，以便于跟踪用户的操作状态。

与 Session 相联系的是 javax.servlet.http.HttpSession 类，Session 是自动创建的。



提示

不同的用户对应的 Session 对象是不相同的。

⊙ pageContext 对象

pageContext 对象用于管理属于 JSP 中特殊可见部分中的已经命名对象的访问。它是 JSP 中的一个新类。

与 pageContext 相联系的是 javax.servlet.jsp.PageContext 类。

⊙ Application 对象

Application 对象用于在多个程序中保存信息。用来在所有用户之间共享信息，并在 Web 应用程序运行期间持久地保持数据。

与 Application 相联系的是 ServletContext 类，通过使用 getServletConfig().getContext()方法得到。一旦创建了 Application 对象，该对象将一直保持下去，直到服务器关闭为止。

提示

每个用户的 application 对象都是相同的，每一个用户都共用同一个 application 对象。

⊙ Config 对象

Config 对象用于配置处理 JSP 程序的句柄，而且只在 JSP 页面范围内合法。

与 Config 相联系的是 javax.servlet.ServletConfig 类。

⊙ page 对象

page 对象代表 JSP 对象本身，或者说代表编译后的 servlet 对象。



与 page 相联系的是 javax.servlet.jsp.HttpJspPage 类。

◉　exception 对象

exception 对象代表 JSP 应用中的错误对象，只有在 JSP 页面的 page 指令中指定 isErrorPage="true"后，才可以在本页面中使用 exception 对象。exception 对象是一个 java.lang.Throwable 对象。

从 6.2 节开始将通过具体的实例讲解各个 JSP 内置对象的使用方法和技巧。

6.2　out 对象

out 对象对应于 jspWriter 类，用于向客户端输出数据。jspWriter 类是一个数据输出流对象，所以 out 对象的主要方法都是与数据流对象相关的，用于把结果输出到网页上。

6.2.1　out 对象常用方法



out 对象的常用方法如表 6-1 所示。

表 6-1　out 对象常用方法

方　法	描　述
void print/println (Boolean｜char｜char[]｜double｜float｜int｜long｜object｜string｜)	输出各种类型的数据
void newLine()	输出一个换行字符
void flush()	输出缓冲区中的数据
void close()	关闭输出流
void clearBuffer()	清除缓冲区中的数据，并把数据输出到客户端
void clear()	清除缓冲区中的数据，但不会把数据输出到客户端
int getBufferSize()	获取以 kb 为单位的目前缓冲区的大小
int getRemaining()	获取缓冲区中没有被占用的空间的大小
boolean isAutoFlush()	返回布尔值表示是否自动输出

6.2.2　out 对象应用实例

下面通过实例来介绍 out 对象的使用。主要的功能是通过 out 对象的 println 方法在客户端浏览器上显示从 out 对象中读取的其他信息，如缓冲区的大小，是否自动输出等设置。

```
//Out.jsp
```

```
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
<title>
out 对象演示
</title>
</head>
<body>
<h2 align="center"> out 对象演示 </h2>
<%
  out.println("信息一<br>");
  out.println("信息二 <br>");
  out.flush();
  out.println("剩余缓冲区大小为:"+ out.getRemaining()+ "字节<br>");
  out.println("默认缓冲区大小为:"+ out.getBufferSize()+ "字节<br>");
  out.println("是否设置 AutoFlush:"+ out.isAutoFlush());
%>
</body>
</html>
```

运行该实例，结果如图 6-1 所示。

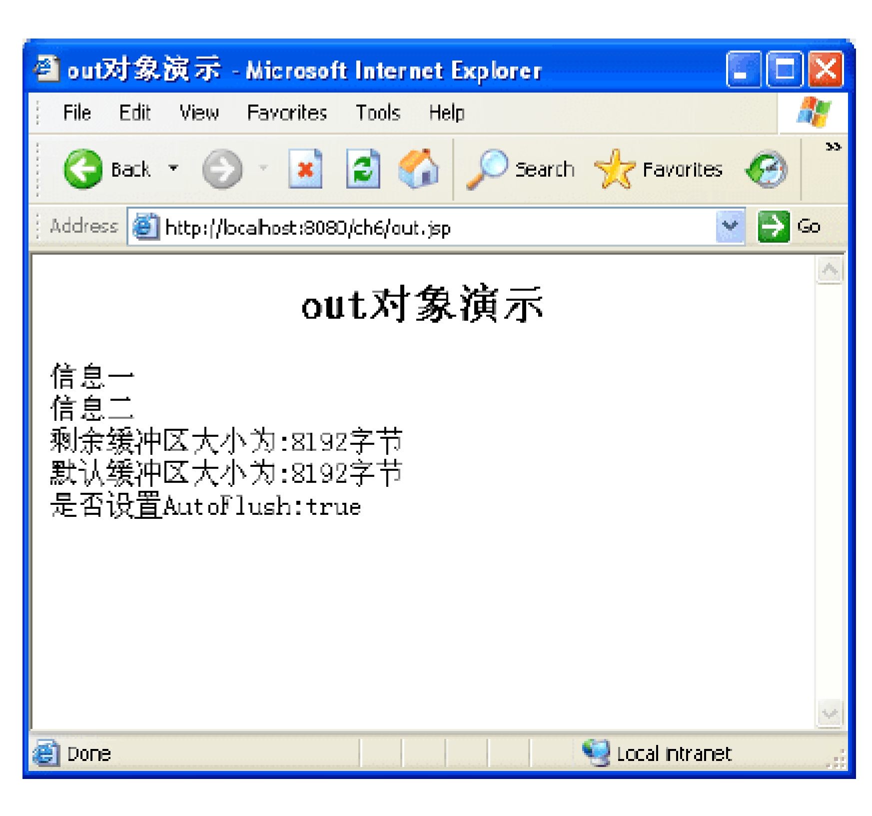
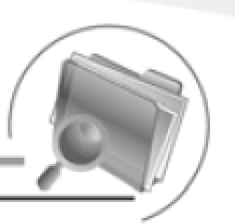


图 6-1　out 对象演示

6.3　request 对象

request 对象封装了客户端请求的所有信息，如请求的来源、标头、cookies 和请求相关的参数等。通过调用该对象的方法可以获取所有这些信息，通过 getParameter(String name)方法可以



获取通过请求传送的所有参数值。从而在各个 JSP 页面之间传送和控制数据。

6.3.1　request 对象常用方法

request 对象的常用方法如表 6-2 所示。

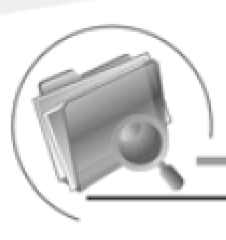
表 6-2　request 对象常用方法

方　法	描　述
Object getAttribute(String name)	返回 name 属性值，该属性不存在时返回 null
Enumeration getAttributeNames()	返回 request 对象的所有属性名称的集合
Enumeration getHeaders(String name)	返回指定 HTTP 头的所有值的集合
String getMethod()	获取客户端向服务器端发送请求的方法(GET、POST)
String getParameter(String name)	获取客户端传送给服务器端的参数值
String getHeader(String name)	获取 HTTP 协议定义的文件头信息
Enumeration getParameterNames()	返回请求中所有参数的集合
String[] getParameterValue(String name)	获取指定参数的所有值
String getProtocol()	获取客户端向服务器端传送数据所依据的协议名称
String getQueryString()	获取查询字符串
String getRequestURI()	获取发出请求字符串的客户端地址
String getRemoteAddr()	获取客户端的 IP 地址
String getRemoteHost()	获取客户端的名字
String getServerName()	获取服务器的名字
String getServletPath()	获取客户端所请求的脚本文件的文件路径
int getServerPort()	获取服务器的端口号
void setAttribute(String name , java.lang. Object objt)	设置名字为 name 的 request 参数的值，该值由 java.lang.Object 类型的 objt 指定

6.3.2　request 对象应用实例

下面的实例将客户端请求的 HTTP 头信息全部读取并显示出来，包括请求方法、URI、协议、远程用户、地址、主机、路径、内容的长度和类型、服务器信息以及客户端浏览器信息等。

```
//Request.jsp:
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
 <title>Request 对象演示</title>
```



```
</head>
<body>
<h2 align="center"> Request 请求信息 </h2>
<hr>
JSP 请求方法：<%= request.getMethod() %> <br>
请求 URI：<%= request.getRequestURI() %> <br>
请求协议：<%= request.getProtocol() %> <br>
Servlet 路径<%= request.getServletPath() %> <br>
远程用户：<%= request.getRemoteUser() %> <br>
远程地址：<%= request.getRemoteAddr() %> <br>
远程主机：<%= request.getRemoteHost() %> <br>
路径信息：<%= request.getPathInfo() %> <br>
内容长度：<%= request.getContentLength() %> <br>
内容类型：<%= request.getContentType() %> <br>
服务器名：<%= request.getServerName() %> <br>
服务器端口：<%= request.getServerPort() %> <br>
<hr>
您使用的浏览器是：<%= request.getHeader("User-Agent") %>
<hr>
</body>
</html>
```

运行该实例，结果如图 6-2 所示。

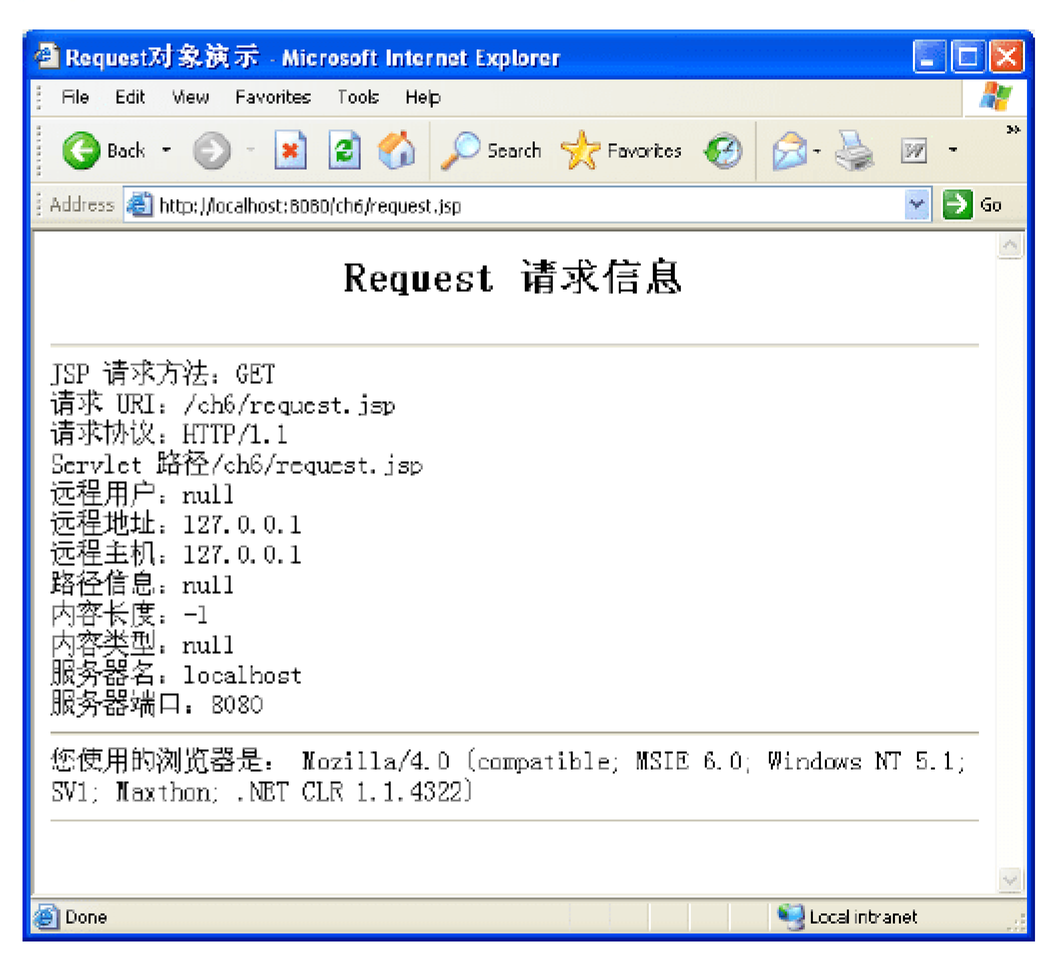


图 6-2　request 对象演示





6.4　response 对象

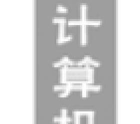
当服务器接收到 request 请求后会对请求做出动态的响应，这个响应的对象就是 response。response 对象主要是将 JSP 容器处理后的结果返回到客户端。

6.4.1　response 对象常用方法

response 对象的常用方法如表 6-3 所示。

表 6-3　response 对象的常用方法

方　法	描　述
void addCookie(Cookie cook)	添加一个 cookie 对象，用来保存客户端的用户信息
void addHeader(String name , String value)	添加 HTTP 文件头信息
boolean containsHeader(String name)	判断指定名字的 HTTP 文件头是否已经存在
void flushBuffer()	强制把当前缓冲区的内容发送到客户端
int getBufferSize()	获取以 kb 为单位的缓冲区大小
void reset()	清空 buffer 中的所有内容
void resetBuffer()	清空 buffer 中所有的内容，但是保留 HTTP 头和状态信息
void sendRedirect(String locationg)	把响应发送到另外一个位置进行处理
void setBufferSize(int size)	设置以 kb 为单位的缓冲区大小
void sendError(int)	发送错误，包括状态码和错误信息
void setHeader(String name , String value)	设置指定名字的 HTTP 文件头的值



6.4.2　response 对象应用实例

该实例使用 response 对象定时刷新页面：

```
//Response.jsp
<%@ page contentType="text/html;charset=GB2312" %>
<%@ page import="java.util.Date" %>
<html>
<head>
<title>定时刷新页面</title>
</head>
<body>
<b> response 对象演示</b>
```

```
<br>
<b>当前时间为：</b>
<% response.setHeader("refresh","10"); %>
<%
out.println(new Date());
%>
</body>
</html>
```

运行该实例，结果如图 6-3 所示。

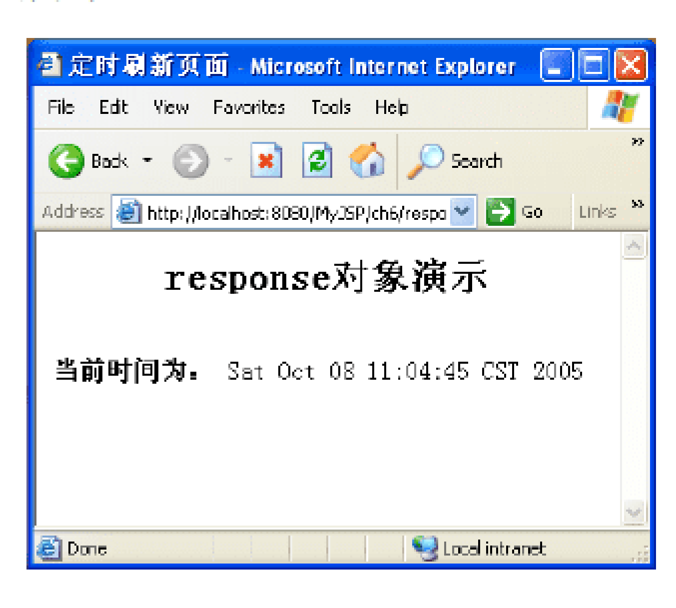


图 6-3　response 对象演示

该页面每 10 秒自动刷新一次，显示时间也随之改变。

6.5　session 对象

session(会话)对象在第一个 JSP 页面被装载时自动创建，完成会话期管理。从一个客户打开浏览器并连接到服务器开始，到客户关闭浏览器离开这个服务器结束，被称为一个会话。当一个客户访问服务器时，可能会在服务器的几个页面之间反复连接或者反复刷新一个页面，服务器应当通过某种办法知道这是同一个客户，这就需要 session 对象。

当一个客户首次访问服务器上的某个 jsp 页面时，jsp 引擎将产生一个 session 对象，同时分配一个 String 类型的 Id 号，jsp 引擎同时将该 Id 号发送到客户端，存放在 Cookie 中，这样 session 对象和客户之间就建立了一一对应关系。当客户再次访问连接该服务器的其他页面时，不用再分配给客户新的 session 对象，直到客户关闭浏览器之后，服务器端才将该客户的 session 对象注销，并且和客户的会话对应关系消失。当客户重新打开浏览器再次连接到该服务器时，服务器将为该客户创建一个新的 session 对象。

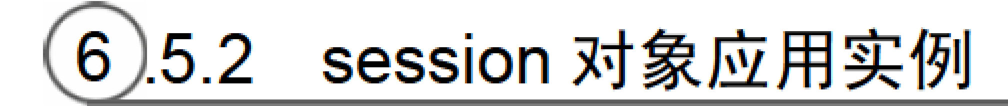
6.5.1　session 对象常用方法

session 对象的常用方法如表 6-4 所示。



表 6-4　Session 对象常用方法

方　　法	描　　述
getAttribute(String name)	获取与指定名字相关联的 session 属性值
getAttributeNames()	获取 session 内所有属性的集合
getCreationTime()	返回 session 的创建时间，最小单位千分之一秒
getId()	获取 session 标识
getLastAccessedTime()	返回与当前 session 相关的客户端最后一次访问的时间，由 1970-01-01 算起，单位为毫秒
getMaxInactiveInterval(int interval)	返回总时间，以秒为单位，表示 session 的有效时间(session 不活动时间)。-1 表示永不过期
getServletContext()	返回一个该 JSP 页面对应的 ServletContext 对象实例
setAttribute(String name, String value)	设置指定名称的 session 属性值
setMaxInactiveInterval(int interval)	设置 session 的有效期
removeAttribute(String name)	移除指定名称的 session 属性
invalidate()	销毁这个 session 对象
isNew()	判断一个 session 是否由服务器产生



6.5.2　session 对象应用实例

该实例演示一个简单的在线购物服务，使用 session 对象来保存选购商品的信息，以保证每个用户只看到自己所选的商品。参看如下代码清单：

```
//session1.jsp

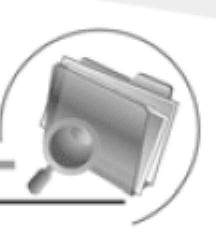
<%@ page contentType="text/html; charset=gb2312" %>
<html>
<head>
<title> Session 对象演示</title>
</head>
<body>
<h2 align="center">使用 session 制作在线购物</h2>
<form action="session2.jsp">
<center>
<table border="1">
<tr bgcolor="lightblue">
<th>商品名称</th>
<th>商品价格</th>
<th>购买数量</th>
```




```
</tr>
<tr bgcolor="yellow">
<td>商品 1</td>
<td>150</td>
<td> <input type="text" name="商品 1"> </td>
</tr>
<tr bgcolor="yellow">
<td>商品 2</td>
<td>1000</td>
<td> <input type="text" name="商品 2"> </td>
</tr>
<tr bgcolor="yellow">
<td>商品 3</td>
<td>535</td>
<td> <input type="text" name="商品 3"> </td>
</tr>
<tr bgcolor="yellow">
<td>商品 4</td>
<td>2340</td>
<td> <input type="text" name="商品 4"> </td>
</tr>
<tr>
<td > <input type="submit" value="提　交"> </td>
<td> <input type="reset" value="重　置"> </td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

上面这段代码是实现让用户选择购买商品以及输入购买数量的窗体，窗体里每一项商品都有一个变量名称：【商品 1】~【商品 4】，用来记录该项商品的购买数量。

```
//session2.jsp
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>购买商品列表</title>
</head>
```



```
<body>
<%
String name,count;
request.getSession(true);
Enumeration goods = request.getParameterNames();
while( goods.hasMoreElements())
{
  name = (String)goods.nextElement();
  count = request.getParameter(name);
  session.putValue(name,count);
}
String names[] = session.getValueNames();
out.print("<font size=5 color=green>商品列表</font> <br>");
for(int i=0; i<names.length; i++)
{
  out.print(new String(names[i].getBytes("ISO8859_1"))+":");
  out.print(session.getValue(names[i])+"<br>");
}
%>
</body>
</html>
```

上面这段程序中首先建立一个 session 对象，接着处理上一个窗体传送过来的数据。request.getParameterNames()是取得窗体所有商品的变量名称，返回的数据类型是 Enumeration，该类型包含在 java.util.*包中，因此在程序一开始必须以<%@ page import="java.util.*" %>将其加载到页面中。

运行 session1.jsp，结果如图 6-4 所示。

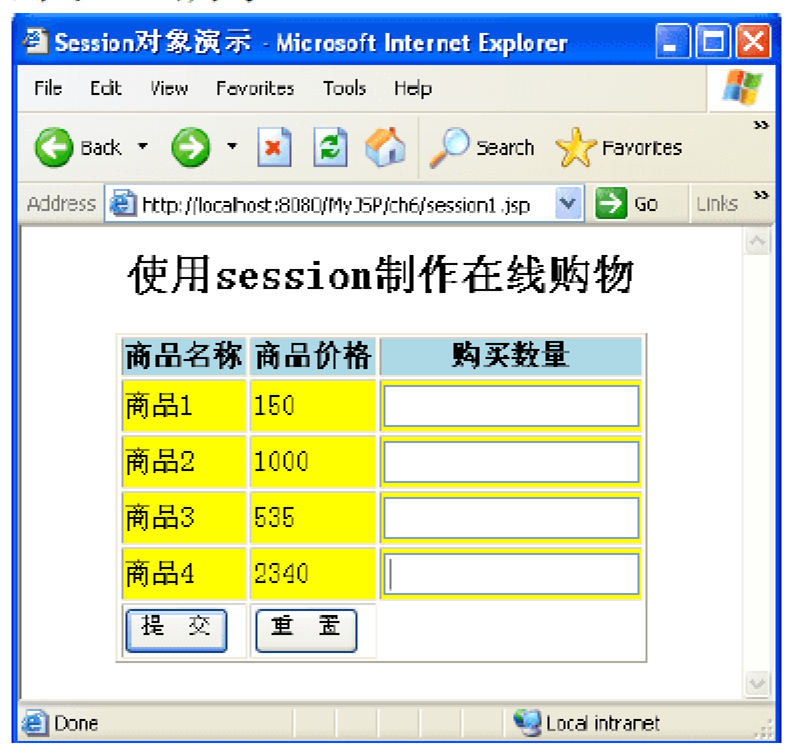


图 6-4　session 对象演示 1

在页面上输入各种商品的购买数量后，单击【提交】按钮，结果如图 6-5 所示。



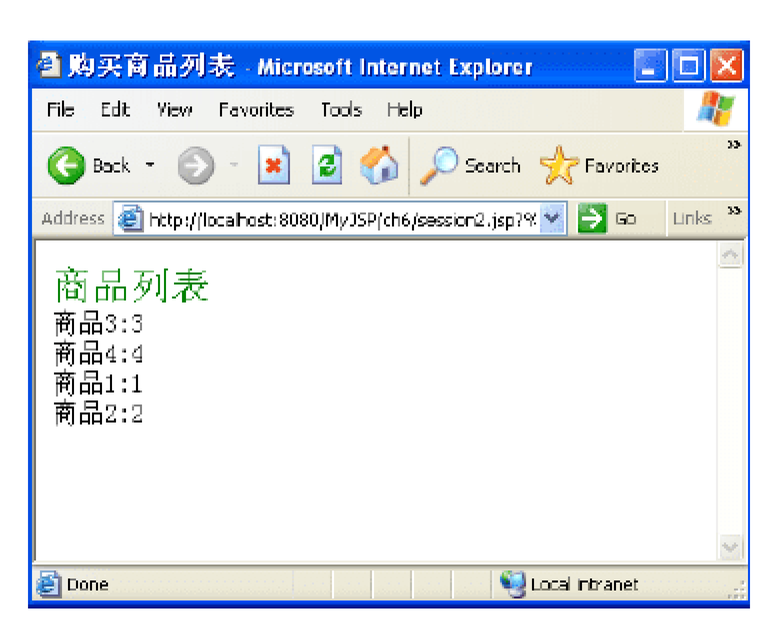


图 6-5　session 对象演示 2

该实例比较简单，只是让用户选购商品，至于当用户送出此购物信息后，服务器端程序还必须计算其购物的总价，或者将数据存入数据库，甚至用户进入在线购物前还必须进行身份验证等工作，读者可以自己进行设计练习。



6.6　pageContext 对象

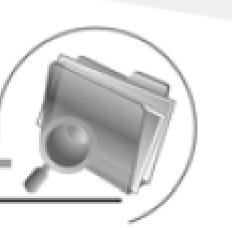
pageContext 对象用于存储本 JSP 页面的相关信息，如属性、内置对象等。

6.6.1　pageContext 对象常用方法

pageContext 对象的常用方法如表 6-5 所示。

表 6-5　pageContext 对象常用方法

方　法	描　述
void setAttribute(String name, Object value, int scope) void setAttribute(String name, Object value)	在指定的共享范围内设置属性
Object getAttribute(String name, int scope) Object getAttribute(String name)	获取指定共享范围内 name 的属性值
Object findAttribute(String name)	按页面、请求、会话和应用程序共享范围搜索已命名的属性
void removeAttribute(String name, int scope) void removeAttribute(String name)	移除指定名称和共享范围的属性
void forward(String url)	将页面导航到指定的 URL
Enumeration getAttributeNamesInScope(int scope)	获取指定共享范围内的所有属性名称的集合
int getAttributeScope(String name)	获取指定属性的共享范围
JspWriter getOut()	获取页面的 out 对象
Object getPage()	获取页面的 page 对象
ServletRequest getRequest()	获取页面的 request 对象



(续表)

方　　法	描　　述
ServletResponse getResponse()	获取页面的 response 对象
ServletConfig getConfig()	获取页面的 config 对象
ServletContext getServletContext()	获取页面的 servletContext 对象
HttpSession getSession()	获取页面的 session 对象
void release()	重置 pageContext 内部状态，释放所有内部引用
void initialize(Servlet servlet, ServletRequest request, ServletResponse response, String errorPageURL, boolean needSession, int bufferSize, boolean autoFlush)	初始化未经初始化的 pageContext 对象

6.6.2　pageContext 对象应用实例



该实例获取所有属性范围为 Application 的属性名称，然后依次显示这些属性：

pageContext.jsp

```
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>
pageContext  对象演示
</title>
</head>
<body >
<h2 align="center"> pageContext  对象演示</h2>
<%
  String appAttrib;
  int count = 0;
  Enumeration attributes = pageContext.getAttributeNamesInScope
                              (pageContext.APPLICATION_SCOPE);
  while(attributes.hasMoreElements())
  {
   count += 1;
   appAttrib = (String) attributes.nextElement();
      out.print("Application  属性" + count +": " + appAttrib + "<br>");
```

```
    }
%>
</body>
</html>
```

运行该实例，结果如图 6-6 所示。

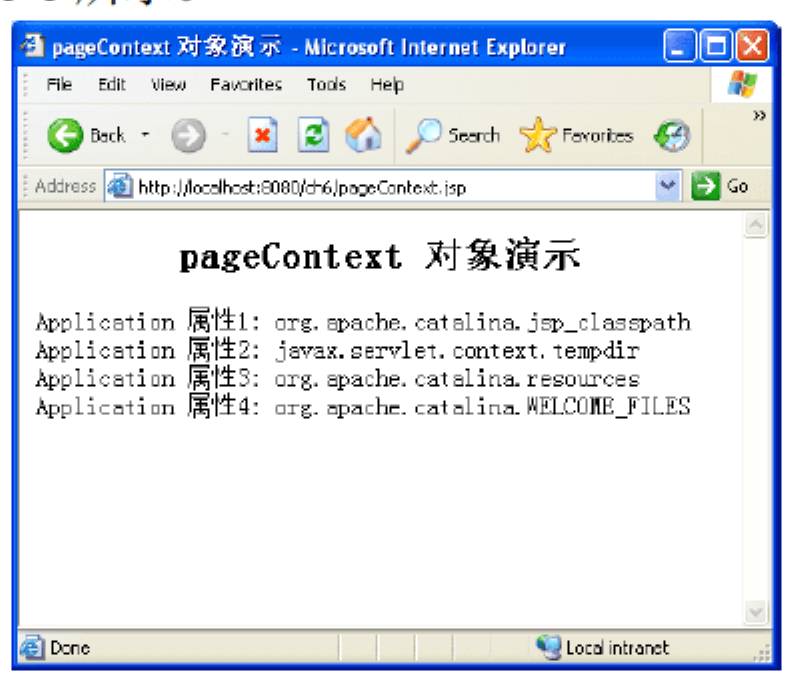


图 6-6 pageContext 对象演示

6.7 application 对象

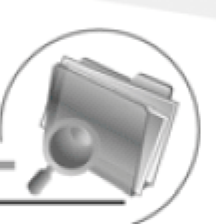
服务器启动后就产生了一个 application 对象，当客户在所访问的网站的各个页面之间浏览时，这个 application 对象都是同一个，直到服务器关闭。但是与 session 不同的是，所有客户的 application 对象都是同一个，即所有客户共享这个内置的 application 对象。application 对象提供了对 javax.servlet.ServletContext 对象的访问。

6.7.1 application 对象常用方法

application 对象的常用方法如表 6-6 所示。

表 6-6 application 对象常用方法

方　法	描　述
Object getAttribute(String name)	返回由 name 指定的 application 属性
Enumeration getAttributes()	返回所有的 application 属性
ServletContext getContext(String uripath)	获取当前应用的 ServletContext 对象
String getInitParameter(String name)	返回由 name 指定的 application 属性的初始值
Enumeration getInitParameters()	返回所有的 application 属性的初始值的集合
String getMimeType(String file)	返回指定文件的类型，未知类型返回 null。一般为"text/html"和"image/gif"



(续表)

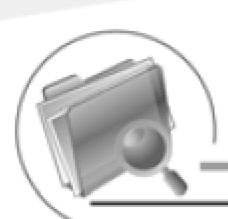
方　　法	描　　述
String getRealPath(String path)	返回给定虚拟路径所对应物理路径
URL getResource(String path)	返回指定的资源路径对应的一个 URL 对象实例，参数要以" / "开头
Set getResourcePaths(String path)	返回存储在 web-app 中所有资源路径的集合
String getServerInfo()	获取应用服务器版本信息
Servlet getServlet(String name)	在 ServletContext 中检索指定名称的 servlet
Enumeration getServlets()	返回 ServletContext 中所有 servlet 的集合
String getServletContextName()	返回本 web 应用的名称
Enumeration getServletContextNames()	返回 ServletContext 中所有 servlet 的名称集合
void log(Exception ex, String msg) void log(String msg, Throwable t) void log(String msg)	把指定的信息写入 servlet log 文件
void removeAttribute(String name)	移除指定名称的 application 属性
void setAttribute(String name, Object value)	设定指定的 application 属性的值

6.7.2　application 对象应用实例

下面的实例定义并取得了 application 中的变量并显示其内容。

application.jsp

```
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>
application  对象演示！
</title>
</head>
<body >
<h2 align="center"> application  对象演示</h2>
<%
   String appName;
   application.setAttribute("老师","李玉");
   application.setAttribute("学生 1","王彦");
   application.setAttribute("学生 2","章晓华");
```



```
    Enumeration names = application.getAttributeNames();
    while(names.hasMoreElements())
    {
        appName = (String) names.nextElement();
        out.print(appName + "是：  ");
        out.print(application.getAttribute(appName) + "<br>");
    }
%>
</body>
</html>
```

运行该实例，结果如图 6-7 所示。

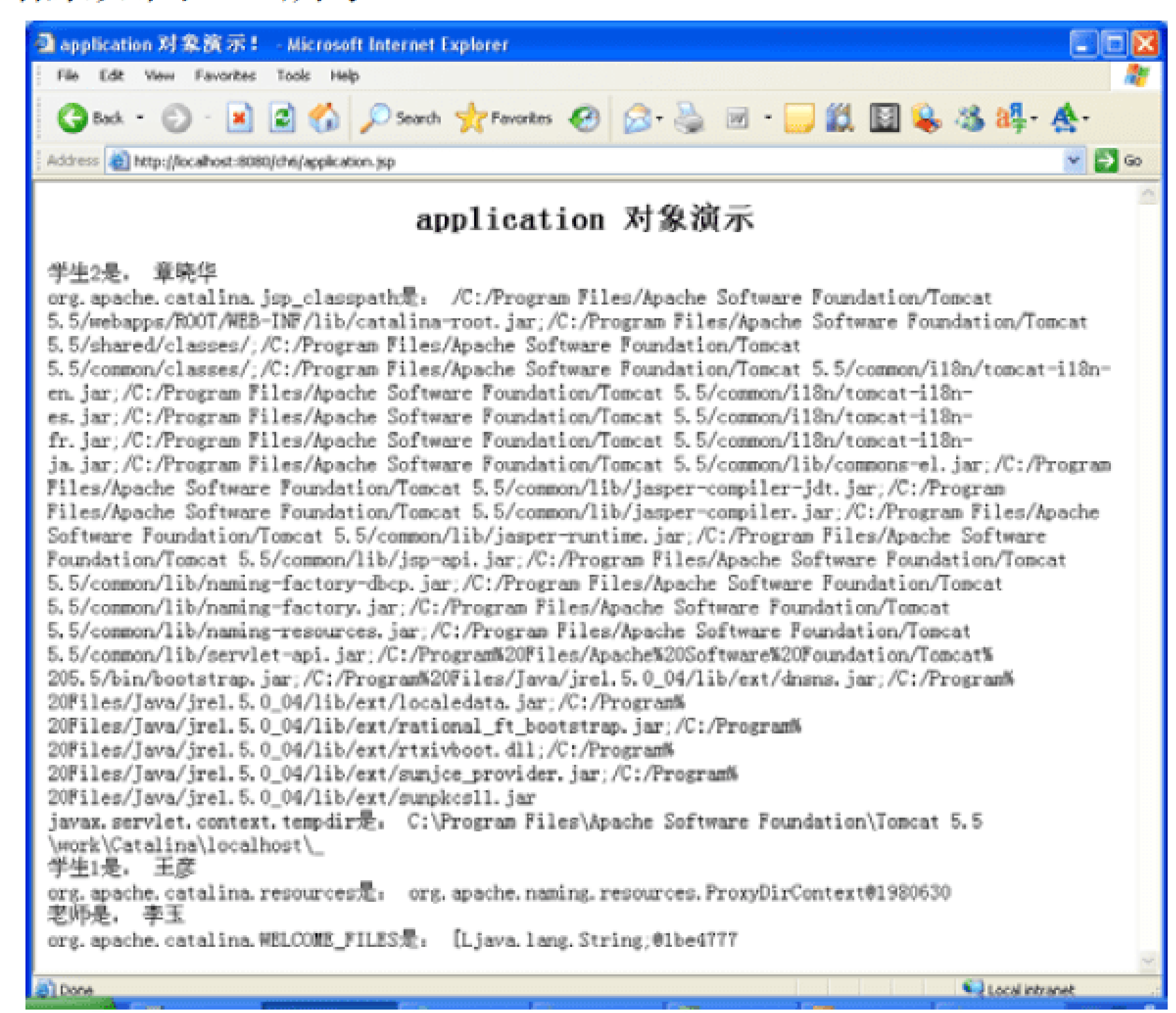


图 6-7 application 对象演示

由执行结果可以发现，在 application 中除了自定义的数据以外还包括其他默认数据。

6.8 config 对象

config 对象用来存放 Servlet 初始的数据结构。

6.8.1 config 对象常用方法

config 对象的常用方法如表 6-7 所示。



表 6-7　config 对象常用方法

方　　法	描　　述
String getInitParameter(String name)	返回名称为 name 的初始参数的值
Enumeration getInitParameters()	返回这个 JSP 所有的初始参数的名称集合
ServletContext getContext()	返回执行者的 Servlet 上下文
String getServletName()	返回 Servlet 的名称

6.8.2　config 对象应用实例

下面通过实例来演示 config 对象的使用。

```
config.jsp
<%@ page contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.*" %>
<html>
<head>
<title>
config 对象演示!
</title>
</head>
<body>
<h2 align="center">config 对象演示</h2>
<%
  Enumeration e = config.getInitParameterNames();
  String paramName;
  String paramValue;
  while(e.hasMoreElements())
  {
    paramName = (String)e.nextElement();
    paramValue = config.getInitParameter(paramName);
%>
参数:<%= paramName %> <br />
参数值:<%= paramValue %> <p>
<%
  }
%>
Servlet name: <%= config.getServletName() %>
</body>
</html>
```



可以在应用程序环境的 web.xml 文件中设置初始参数，代码如下：

```
<servlet>
        <servlet-name>config</servlet-name>
        <jsp-file>/ch6/config.jsp</jsp-file>
        <init-param>
          <param-name>test</param-name>
          <param-value>It's just a test!</param-value>
        </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>config</servlet-name>
        <url-pattern>/ch6/config.jsp</url-pattern>
</servlet-mapping>
```



运行该实例，结果如图 6-8 所示。

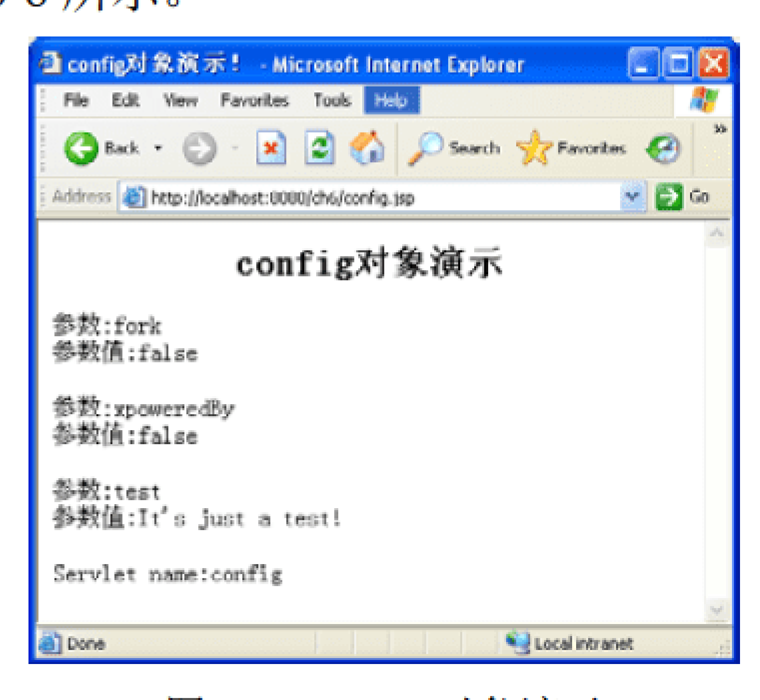


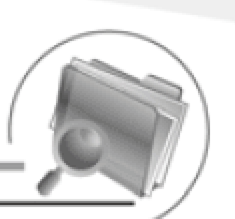
图 6-8　config 对象演示

6.9　page 对象

page 对象代表 JSP 对象本身，或者说代表编译后的 Servlet 对象，可以用((javax.servlet.jsp.HttpJspPage)page)来获取它的方法和属性，包括所有 Servlet 类所定义的方法。不过在实际应用中，page 对象很少被使用。

下面来看一个简单的实例：

```
//page.jsp
<%@ page   info="该实例为简单的 page 对象演示!"
            contentType="text/html; charset=gb2312" %>
<html>
<head>
<title>page 对象演示</title>
```



```
</head>
<body>
<h2 align="center"> page 对象演示</h2>
page info: <%=((javax.servlet.jsp.HttpJspPage)page).getServletInfo() %>
</body>
</html>
```

在这个实例中，首先设定 page 指令的 info 属性为【该实例为简单的 page 对象演示！】，page 对象的类型为 java.lang.Object，然后调用 javax.servlet.jsp.HttpJspPage 中的 getServletInfo() 方法，将 info 信息打印出来，执行结果如图 6-9 所示。

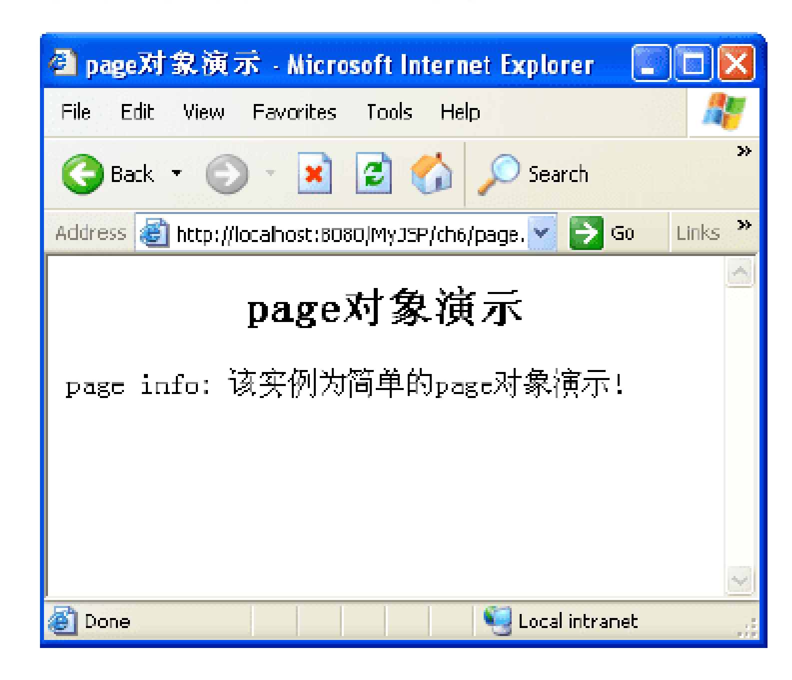
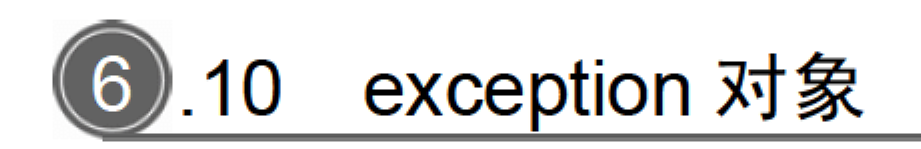


图 6-9　page 对象演示



6.10　exception 对象

exception 对象提供了对出错的 JSP 页面内的异常访问。只有在 JSP 页面的 page 指令中指定 isErrorPage="true"之后，才可以在本页面中使用 exception 对象。

6.10.1　exception 对象常用方法

exception 对象的常用方法如表 6-8 所示。

表 6-8　exception 对象常用方法

方　法	描　述
Throwable fillInStackTrace()	将当前 stack 信息记录到 exception 对象中
String getLocalizedMessage()	获取本地语系的错误提示信息
String getMessage()	获取错误提示信息
StackTrackElement[] getStackTrace()	返回对象中记录的 call stack track 信息
Throwable initCause(Throwable cause)	将另外一个异常对象嵌套进当前异常对象中
Throwable getCause()	获取嵌套在当前异常对象中的异常



(续表)

方　法	描　述
void printStackTrace() void printStackTrace(printStream s) void printStackTrace(printWriter s)	打印出 Throwable 及其 call stack trace 信息
void setStackTrace(StackTraceElement[] stackTrace)	设置对象的 call stack trace 信息

6.10.2 exception 对象应用实例

下面的实例演示了使用 exception 对象的方法。exception.jsp 文件是一个错误页面，它负责其他页面的错误处理工作。throwException.jsp 文件是为了演示而特意制造了一个 exception，并在其 page 指令中指定错误页面为 exception.jsp。


```
//exception.jsp
<%@ page isErrorPage="true" import="java.io.*" %>
<html>
  <body>
    <font color="red">
      <%= exception.toString() %><br>
      <%
        exception.printStackTrace(new PrintWriter(out));
      %>
    </font>
  </body>
</html>
//throwException.jsp
<%@ page errorPage="exception.jsp" %>
<html>
  <body>
    <%
      int i= 100/0;
    %>
  </body>
</html>
```

运行上述程序之后，throwException.jsp 由于抛出了异常被导向 exception.jsp 页面，在 exception.jsp 页面中处理抛出的异常(这里是打印异常的详细信息)，运行结果如图 6-10 所示。



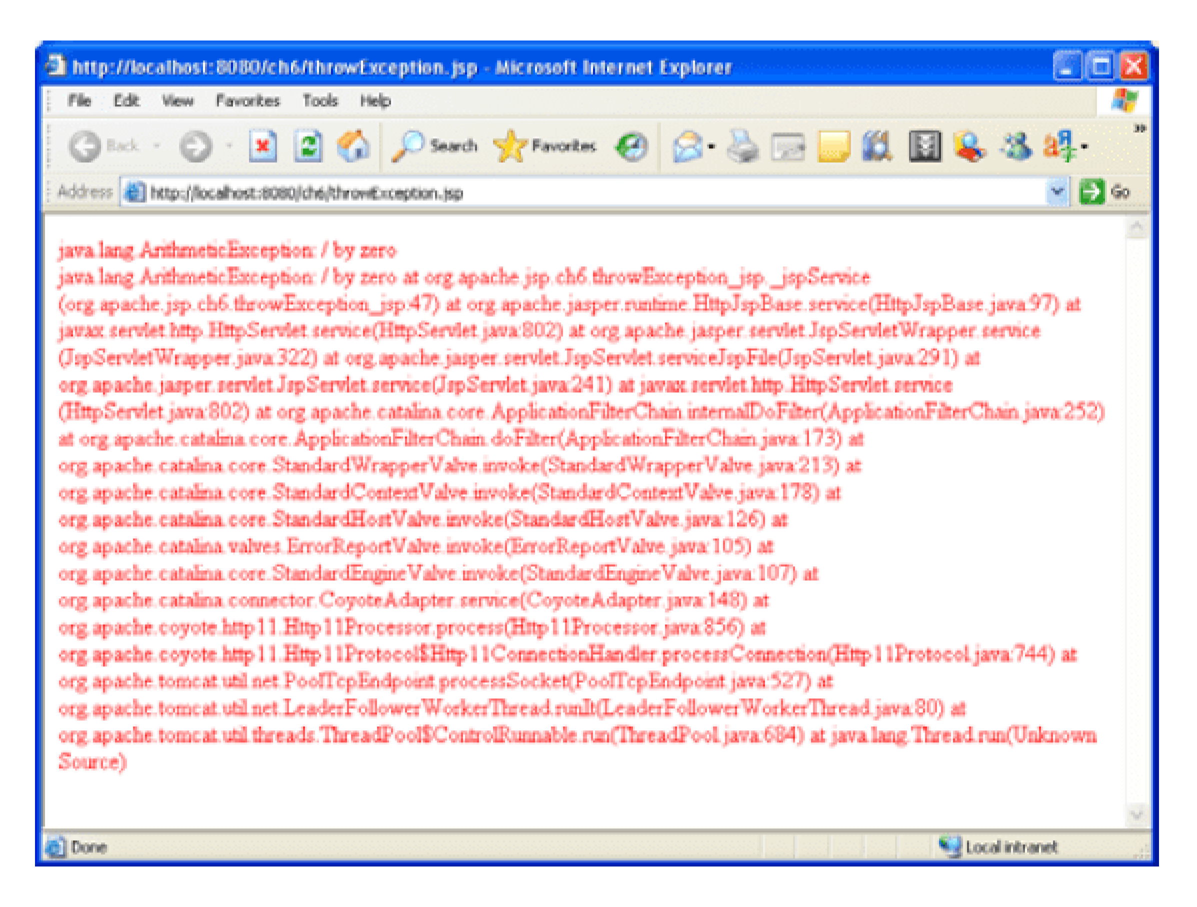
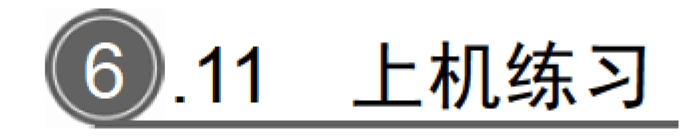


图 6-10　exception 对象演示



6.11　上机练习



本章上机实验主要练习如何使用 JSP 的各种内置对象。其中重点掌握如何使用 out、request、response、session、pageContext、application、config、page 和 exception 对象的常用方法。

下面以在介绍 session 对象时的实例为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，将打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮后 Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中会出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Other】命令，弹出 New 对话框新建 JSP 文件。

(7) 在 New 对话框中选择【Web】|【JSP】选项，单击 Next 按钮弹出 New JavaServer Page 对话框。

(8) 在 New JavaServer Page 对话框的 File name 文本框中输入文件名，如 session1.jsp。

(9) 单击 Finish 按钮即可新建 JSP 页面文件。

(10) Eclipse 会创建一个包括了基本的页面框架的新文件。在编辑窗口中输入 session1.jsp 页面的代码。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 重复步骤(6)~(12)创建并检查 session2.jsp 文件。

(14) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(15) 打开一个 Internet Explorer 浏览器，输入对应的 URL，如：http://localhost:8080/test Tomcat/ch6/session1.jsp，观察页面是否正常运行。

6.12 习题

6.12.1 填空题

1. JSP 的内置对象包括________、________、________、________、________、________、________、________和________。

2. Response 对象的来源是________。

3. 对于每个用户都共享同一个对象的是________对象，而每个用户分别使用不同对象实例的是________对象。

6.12.2 选择题

1. 下面哪个方法不属于 session 对象(　　)。

A. getAttributeNames()　　B. getServletContext()

C. invalidate()　　D. addCookie(Cookie cook)

2. 下面哪个方法不属于 request 对象(　　)。

A. getServerName ()　　B. getServerInfo ()

C. getServletPath ()　　D. getServerPort()

3. 能够获取当前页信息并调用页面方法的对象是(　　)。

A. request　　B. page

C. pageContext　　D. session

6.12.3 问答题

1. out.clear 与 out.flush 有什么区别？

2. 如何使用 request 对象获取请求参数？

3. pageContext 可以获取哪些内置对象？



第7章 JSP 与 JavaBean

学习目标

JSP 作为一个很好的动态网站开发语言得到了越来越广泛的应用，在各类 JSP 应用程序中，JSP + JavaBean 的组合成为一种最常见的 JSP 程序标准。

本章先阐述 JavaBean 的工作原理，接着阐述 JavaBean 在 JSP 下的特定语法，最后演示两个使用 JSP+JavaBean 的简单示例。

本章重点

- Java Bean 分类
- Java Bean 的属性
- Java Bean 持久化
- JSP 上的 Java Bean

7.1 JavaBean 简介

JavaBean 是描述 Java 的软件组件模型，有点类似于 Microsoft 的 COM 组件。在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，通过 JavaBean 的组合可以快速生成新的应用程序。对于程序员来说，最好的一点就是 JavaBean 可以实现代码的重复利用，另外，对于程序的易维护性等也有很重大的意义。

JavaBean 通过 Java 虚拟机(Java Virtual Machine)可以得到正确的执行，运行 JavaBean 最低的需求是 JDK1.1 或以上版本。

传统的 JavaBean 应用在可视化的领域，如 AWT 下的应用。自从 JSP 诞生之后，JavaBean 更多地应用在了非可视化领域，在服务器端应用方面表现出了越来越强的生命力。在这里，主

要讨论的是非可视化的 JavaBean，可视化的 JavaBean 在市面上有很多 Java 书籍都有详细的阐述，本书就不作重点介绍了。

7.1.1 非可视化的 JavaBean

非可视化的 JavaBean，顾名思义就是没有 GUI 界面的 JavaBean。在 JSP 程序中常用它来封装事务逻辑、数据库操作等，可以很好地实现业务逻辑和前台程序(如 JSP 文件)的分离，使得系统具有更好的健壮性和灵活性。

比如一个购物车程序，要实现在购物车中添加一件商品的功能，就可以写一个购物车操作的 JavaBean，建立一个 public 的 AddItem 成员方法，在前台 JSP 文件里面直接调用这个方法来实现。如果后来又考虑添加商品时需要判断库存是否有货物，没有货物则不得购买，这时就可以直接修改 JavaBean 的 AddItem 方法，加入处理语句来实现，这样就完全不用修改前台 JSP 程序了。

当然，也可以把这些处理操作完全写在 JSP 程序中，不过这样的 JSP 页面可能就有成百上千行代码了，光看代码就是一个头疼的事情，更不用说修改了。由此可见，通过 JavaBean 可以很好地实现逻辑的封装、程序的易于维护等。

如果使用 JSP 开发程序，一个很好的习惯就是多使用 JavaBean。

7.1.2 DataBean 和 ActionBean

非可视化 Bean 分为 DataBean 和 ActionBean 两大类。DataBean 是存储必要数据的 Beans，而 ActionBean 是运用 DataBean 上存储的数据进行特定作业的 Beans。

一个简单的例子，实现电子邮件发送的简单应用，用于存储发送邮件数据的是 DataBean，针对这些数据为基础完成邮件发送任务的是 ActionBean。

将 JavaBeans 分为 DateBean 和 ActionBean 以后，对代码的修改和网站的维护管理都变得非常便利。仍然以发送电子邮件为例，当需要改动收件人信息时，不需要改动 ActionBean 部分的代码，只需要修改 DataBean 的收件人参数即可。

反过来，如果想修改发送邮件的服务器、或者需要实现在每封信的末尾添加广告等功能，就不需要更改 DataBean，直接修改 ActionBean 即可。

7.1.3 ParameterBean 和 DatabaseBean

如果将 DataBean 做进一步分类，可以根据 JSP 程序中输入 DataBean 的数据来源，将 DataBean 分为 ParameterBean 和 DatabaseBean。ParameterBean 是存储用户提交数据的 Bean，而 DatabaseBean 是存储在数据库中数据的 Bean。



 提示

在实际编程过程中会出现 ParameterBean 和 DatabaseBean 的界限非常模糊的情况，一个 Bean 很可能既是 ParameterBean 又是 DatabaseBean。

7.1.4　Beans 的用法

前面已经讲到，JavaBean 是一种组件技术，所以 JavaBean 将内部的动作封装起来，用户看不到它的运行机制，它只提供最小限度的属性接口供外壳控制应用。JavaBean 为了提供组件功能，必须满足以下 3 种条件。

 必须拥有无参数构造函数

JavaBean 要有无参数构造函数，是指在利用 new 生成新类时不必提供特别的参数即可产生类。

例如：

```
String example = new String();
```

在产生 example 这个字符串时，没有必要指定特殊的参数，直接使用 String()就可以产生字符串，这就是无参数构造函数(有参数是指像 string(Bean)这样在括号内提供参数值的情况)。

无参数构造函数是 JSP 容器在自动使用 JavaBean 的时候参考的构造函数，即，当要接收用户 ID 和密码来认证用户，并让用户登录时，将用户输入的信息存储到名为 LoginBean 的 Beans 里面进行处理，在这种情况下，JSP 容器将自动生成 LoginBean 的实体。此时，JSP 容器就是使用 LoginBean 的无参数构造函数生成的实体，再根据用户输入值变换实体的。

 提示

如果没有无参数构造函数，就无法生成实体，并产生错误。但是 JavaBean 类没有特别指定构造函数时，将自动生成无任何作用的无参数构造函数。所以即使不做任何处理上述的条件也可以满足。但是为了养成良好的编程习惯，最好编制无参数构造函数。

如果以 ExampleBean 为名制作一个 Beans 的话，可以编制如下无参数构造函数(在本例中生成的无参数构造函数里面，可以将存储姓名的变量 name 设定为 riceman)：

```
import java.io.*;
public class ExampleBean{
        private String name;
        public ExampleBean(){ //无参数构造函数
                name = "riceman"; //指定名称
}
```



⊙ 编制 Serializable Interface

JavaBean 必须编制 Serializable Interface。这意味着 JavaBean 在存储或传递时非常便利。

在 Java 里面可以将 JavaBean 类以当前状态存入文件中，而且可以通过网络传递给别的机器。Java 的这一功能称为 Serializable。为了使 Serializable 成为可能，必须在相应的类上编制 Serializable Interface。也就是说，如果对 JavaBean 编制了 Serializable Interface，那么在保存或向网络传递某种状态时会比其他技术更为便利。

为了应用以上事项，必须对所有的 JavaBean 编制 Serializable Interface，如下所示。

```
import java.io.*;
public class ExampleBean implements Serializable{
        private String name;
        public ExampleBean(){ //无参数构建函数
                name ="ricename"; //指定名称"ricename"
        }
}
```

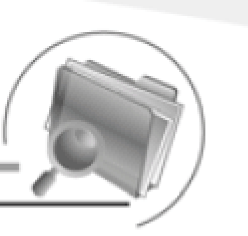

但是在 JSP 中使用到的 JavaBean，就不必声明 Serializable 也可以正常运行。所以 Serializable Interface 关键词可以省略。当然如果想严格遵守 JavaBean 的规定，声明 Serializable 也可以。

⊙ 必须拥有 Property Interface

如果上面提到的两个条件都已经具备的话，JavaBean 就已经具备了作为 JavaBean 所需要的基本特性。但是，还不能说已经全部具备了 JavaBean 的特性。虽然它已经拥有了作为 JavaBean 组件的条件，但是还没有给它设定操作 JavaBean 属性的方法。即如果利用上面的两个特性制作 Bean 的话，制作出来的可能是“没有调频的收音机”或者“没有开关的台灯”。所以必须给 JavaBeans 提供控制属性的接口。

JavaBean 的属性接口使用以“get…”、“set…”为开头的方法来执行其操作，向上个例子中所说的控制 name 变量的方法就是 getName()和 setName()，添加了 name 属性的属性接口的 ExampleBean 的代码如下所示：

```
import java.io.*;
public class ExampleBean implements Serializable{
        private String name;
        public ExampleBean(){ //无参数构建函数
                name = "ricename"; //指定名称"ricename"
        }
        public String getName(){ //getName 函数
                return name;
        }
        public String setName(String name){ //setName 函数
                this.name = name;
        }
}
```



如果想读取 name，又不能更改它的值的话，即把 name 设定为只读，就不要编写 setName()方法。相反，如果想让它可以被修改，但是不能提取的话，就不要编写 getName()方法。

像这样限制使用 setName()和 getName()方法的情况非常多见。编写的 JavaBean 让其他程序使用该组件的话，这个组件的编写者可能希望自己的名字写入到该 JavaBean，此时，为了不让别人更改该属性，就不能提供 setAuthor()方法，而只提供 getAuthor()方法。密码的存储则与此相反。即如果开发一个在内部处理密码的 JavaBean 组件的话，就应该把方法设定成只能输入密码，不能显示密码。所以应提供 setPassword()方法，而不提供 getPassword()方法。

7.1.5　JavaBean 的属性

JavaBean 的属性与一般 Java 程序中所指的属性，或者说与所有面向对象程序设计语言中的对象属性是一个概念，在程序中的具体体现就是类中的变量。在这里主要介绍 JavaBean 的 Simple 和 Index 属性。



- Simple 属性

Simple 属性表示伴随有一对 get/set 方法(C 语言的过程或函数在 Java 程序中称为“方法”)的变量。属性名与和该属性相关的 get/set 方法名对应。例如，如果有 setX 和 getX 方法，则暗指有一个名为 X 的属性。

例如，下面的程序：

```
public class Test1
{
  //属性名为 MyString，类型为字符串
private String MyString;
public Test1()
{
MyString =   "Hello";
}
// set 属性
public void setMyString (String newString)
{
  MyString = newString;
}
//get 属性
public String getMyString ()
{
  return MyString;
}
}
```

◉ Index 属性

Index 属性表示一个数组值。使用与该属性对应的 get/set 方法可以取得数组中的数值。该属性也可以一次设置或获取整个数组的值。例如：

```
public class Test2
{
// dataSet 是一个 index 属性
private int[] dataSet;
public Test2()
{
dataSe t = {1,2,3,4,5,6};
}
//set 整个数组
public void setDataSet(int[] x)
{
   dataset = x;
}
//set 数组中的单个元素值
public void setDataSet(int index, int x)
{
   dataSet[index] = x;
}
//get 整个数组值
public int[] getDataSet()
{
   return dataSet;
}
//get 数组中的指定元素值
public int getDataSet(int x)
{
   return dataSet[x];
}
}
```

7.1.6 JavaBean 的持久化

当 JavaBean 在构造工具内被用户化，并与其他 Beans 建立连接之后，它的所有状态都应当被保存，下一次被装载进构造工具内或在运行时，就应当是上一次修改完的信息。为了做到这一点，就要把 Bean 的某些字段信息保存下来，在定义 Bean 时使它实现 java.io.Serializable 接口。例如：



```
public class Button implements java.io.Serializable
{
…
}
```

实现了序列化接口的 Bean 中的字段信息将被自动保存。若不想保存某些字段的信息则可以在这些字段前加上 transient 或 static 关键字：transient 和 static 变量的信息是不可以被保存的。通常，一个 Bean 的所有公开出来的属性都应当是被保存的，也可以有选择地保存内部状态。Bean 开发者在修改软件时，可以添加字段，移走对其他类的引用，改变字段的 private/protected/public 状态，这些都不影响类的存储结构关系。然而，当从类中删除一个字段、改变一个变量在类体系中的位置、把某个字段改成 transient/static 或原来是 transient/static 现改为别的特性时，都将引起存储关系的变化。

JavaBean 组件被设计出来之后，一般是以扩展名.jar 的 zip 格式文件存储，在 jar 中包含与 JavaBean 有关的信息，并以 MANIFEST 文件指定其中的哪些类是 JavaBeans。以 jar 文件存储的 JavaBeans 在网络中传送时极大地减少了数据的传输数量，并把 JavaBeans 运行时所需要的一些资源捆绑在一起。

7.1.7　用户化

JavaBean 开发者可以给一个 Bean 添加定制器(Customizer)、属性编辑器(PropertyEditor)和 BeanInfo 接口来描述 Bean 的内容。Bean 的使用者可以在构造环境中通过与 Bean 附带在一起的这些信息来用户化 Bean 的外观和应做的动作。一个 Bean 不必都有 Customizer、PropertyEditor 和 BeanInfo，根据实际情况，这些是可选的。当有些 Bean 较复杂时，就要提供这些信息，以 Wizard 的方式使 Bean 的使用者能够定制一个 Bean。有些简单的 Bean 可能没有这些信息，则构造工具可以使用自带的透视装置，透视出 Bean 的内容，并把信息显示到标准的属性表或事件表中供使用者定制 Bean。前面提到的 Bean 的属性、方法要以一定的格式命名，主要的作用就是供开发工具对 Bean 进行透视。当然也是给程序员在写程序中使用 Bean 提供方便，使其能观其名即知其意。有关用户化的信息，读者可以查阅相关资料以获得更详细的描述。

7.2　JSP 上的 JavaBeans

在 JSP 中提供对 JavaBean 的支持，可以通过操作指令<jsp:useBean>、<jsp:setProperty>以及<jsp:getProperty>来操作 JavaBeans。

首先用<jsp:useBean>定义要应用的 JavaBeans，然后用<jsp:setProperty>来存储属性值，最后用<jsp:getProperty>提取存储的属性值，如图 7-1 所示。

<jsp:useBean>

生成或者调用
JavaBeans对象

<jsp:getProperty> JavaBeans <jsp:setProperty>

调用JavaBeans
内的变量

设置JavaBeans
内的变量

图 7-1 <jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>标记的作用



7.2.1 <jsp:useBean>操作

<jsp:useBean>操作用于创建一个 Bean 实例并指定它的名字、类型和作用范围等。

要使用 Beans，首先要用<jsp:useBean>调用相应的 Bean。如果要调用的 Bean 不存在，就产生一个新的 Bean，如果已经存在就调用这个 Bean。

<jsp:useBean>的语法格式如下：

```
<jsp:useBean id="名字" scope="范围" class="类名称" type="类的种类"
beanType="Bean 的类型"
……>
```

<jsp:useBean>操作的语法格式有两种，如果该操作不包含主体，其语法格式为：

```
<jsp:useBean id="name" scope="scope"    typespec   />
```

如果该操作包含主体，则其语法格式为：

```
<jsp:useBean id="name" scope="scope"   typespec   >
    操作主体部分
  </jsp:useBean>
```

这里 id、scope 和 typespec 属性的含义如下。

◉ id 属性

id 属性用作指定范围内 JavaBean 对象的标识符，在 JSP 页面内声明为一个 Java 脚本变量。该值是一个脚本变量，大小写敏感，必须符合 Java 标识符的命名规则。该值也用在<jsp:setProperty>和<jsp:getProperty>操作的 name 属性中，指定所访问的 JavaBean。

◉ scope 属性

scope 属性指定 JavaBean 存在的范围。它与 PageContext 对象获得的范围相同，可能的值



如下：

- ⊙ page：可以在包含<jsp:useBean>元素的 JSP 文件以及此文件中的所有静态包含文件中使用 Bean，直到页面执行完毕并向客户端发回响应或转到另一个文件为止，该值为默认值。
- ⊙ request：在任何执行相同请求的 JSP 文件中使用 Bean，直到页面执行完毕并向客户端发回响应或转到另一个文件为止。
- ⊙ session：从创建 Bean 开始，就能在任何使用相同 session 的 JSP 文件中使用 Bean。这个 Bean 存在于整个 Session 生存周期内，任何分享此 Session 的 JSP 文件都能使用同一个 Bean。



提示

> 在创建 Bean 的 JSP 文件中，必须在<%@　page %>指令中指定 session=true。

- ⊙ application：从创建 Bean 开始，就能在任何使用相同 application 的 JSP 文件中使用 Bean。这个 Bean 存在于整个 application 生存周期内，任何在分享此 application 的 JSP 文件都能使用同一个 Bean。

上述范围内的 JavaBean 都可以使用 pageContext 对象的 getAttribute 和 setAttribute 访问。在 request、session 和 application 范围内的 JavaBean 可以分别在 Servlet 中使用 ServletRequest、HttpSession 和 ServeletContext 对象的 getAttribute 和 setAttribute 访问。

- ⦿ typespec 类型规范

类型规范由 class、type 和 beanName 属性组合而成。这些属性的组合可以灵活地实现<jsp:useBean>操作，从而定义生成 Bean 的对象和类型。

class 是 Bean 的对象名称。在显示要使用的 Bean 对象时使用 class，默认值为 Null，而且不能缺少；type 的作用是定义 Bean 对象的类型。如果要把名为 family.Mother 的 Bean class 变换为名称为 family.Sister 的 Bean(这个过程称为 casting)，在 useBean 操作指令中可以输入如下代码。type 参数的默认值和 class 参数值相同。

```
<jsp:useBean .. class ="Family.Mother" type="Family.Sister" .. />
```

使用和 class 参数相似的概念定义 Bean 时，如果不使用 class 参数，就要定义 beanName 参数，这是为了完成 Bean 的内部代码而使用的属性。如果定义为 beanName="Family.Mother"，那么使用 useBean 生成的 Bean 的名称就是 Family.Mother。

typespec 的 3 种参数不能同时使用，使用方法有以下 4 种：

```
Class="package.class";
```

只定义输入到 Bean 中的类，type 的默认值和 class 参数相同。

```
Type="package.class";
```

这是在定义类之后，为重新定义 type 而使用的。



```
Class="package.class" type="package.class"
```

作为决定类(class)和 type 时使用，如果 class 和 type 的设定相同，就可以省略 type 参数。

```
beanName=" {package.class, <%=expression %> }" type= "package.class"
```

这个表达式的作用是定义指定 bean-class 的 Bean 名称和 type 参数。beanName 参数可以定义 Bean 的类名称，也可以利用表达式定义 Bean 的类名称。

7.2.2 <jsp:setProperty>操作

<jsp:setProperty>操作用于设置 JavaBean 中的属性值。在使用<jsp:setProperty>之前，必须先用<jsp:useBean>创建 JavaBean。其语法格式如下：




```
<jsp:setProperty name="Name"
          { property= "*" |
            property="propertyName"   param="parameterName"   |
            property="propertyName" value="{string | <%= expression %>}"
          }
  />
```

下面描述各个属性及其用法：

- name 属性

该属性表示已经在<jsp:useBean>中创建的 JavaBean 实例的名字。name 属性值应当与<jsp:useBean>操作中的 id 属性值一致。

- property= "*"

用 name 属性标识了 JavaBean 之后，必须指定要设置的 JavaBean 中的属性名，这就是 property 属性。若该属性的值为"*"，那么此 JavaBean 中的属性列表与当前请求中的参数列表比较，一旦出现匹配，则使用相应的请求参数调用 set 方法。如果 Request 对象的参数值中有空值，那么对应的 JavaBean 属性将不会设定任何值。同样，如果 Bean 中有一个属性没有与之对应的 Request 参数值，那么该属性也不会被设定。

- property="propertyName"   param="parameterName"

使用 Request 中的一个参数值来指定 JavaBean 中的一个属性值。在这个语法中，property 指定 JavaBean 的属性名，param 指定 Request 中的参数名。

- property="propertyName" value="{string | <%= expression %>}"

使用指定的值来设定 JavaBean 的属性，这个值可以是字符串，也可以是表达式。

<jsp:setProperty>的应用有以下 3 种方式：

- <jsp:setProperty name="beanname" property="*" />



- <jsp:setProperty name="beanname" property="request 参数" /> 或者<jsp:setProperty name="beanname" property=" bean 参数" param=" request 参数"/>
- <jsp:setProperty name="beanname" property=" bean 参数" value=" 参数值"/>

第 1 种方式<jsp:setProperty name="beanname" property="*" />的作用是将前一页传递过来的参数存储到 Bean 的参数中。假定设置如下程序段：

```
<jsp:setProperty name="numguess" property="*" />
```

如果像上面的程序那样设置的话，JSP 容器将查找是否存在与 request 相对应的 set 方法，如果有，就将对应的参数值赋值给此参数。如果前一页向本页传递过来两个变量的话，JSP 容器首先查找两个相应的 set 方法是不是在 NumGuess Bean 中已经定义了，如果有，就运行这两个 set 方法。

第 2 种方法不同于第一种方法，它应用于只将 request 得到的参数中的某一个值传递给 Beans 的情况。



如果上一页 request 得到的参数和 Bean 的变量同名，那么为了将 request 的参数值传递给 Bean，就使用<jsp:setProperty name="beanname" property="request 参数" />；假如上一页 request 中的参数和 Bean 的变量名不一样，要将 request 中的参数值传递给 Bean，就要用<jsp:setProperty name="beanname" property=" bean 参数" param=" request 参数"/>。

第 3 种方式是利用 value 的方式。这是向 Bean 的变量直接传递的方式，假如有如下的 JSP 代码：

```
<jsp:setProperty name ="numguess" property="guess" value="4" />
```

将这个代码转换为 Servlet 代码后如下所示：

```
JspRuntimeLibrary.introspecthelper(pageContext.findAttribute("numguess"), "guess", "4",null,null,false);
```

7.2.3 <jsp:getProperty>操作

<jsp:getProperty>操作对应于设置属性的<jsp:setProperty>操作，它将获得 JavaBean 的属性值，并将其使用或显示在 JSP 页面中。在使用<jsp:getProperty>之前，必须先使用<jsp:useBean>创建 JavaBean。其语法格式如下：

```
<jsp:getProperty name="name" property="propertyName" />
```

<jsp:getProperty>操作包含 name 和 property 两个属性。name 属性指定 JavaBean 实例的名称，该值必须与<jsp:useBean>中的 id 属性值一致；property 属性则指定要显示其值的 JavaBean 属性名称。

getProperty 的情况比较简单，因为只需要从 Bean 中将值提取出来即可，所以只有一种调用方式：

```
<jsp: getProperty name="Bean 的名称" property="参数" />
```

例如有如下的标记：

```
<jsp:getProperty name ="numguess" property="success" />
```

它将转换为 Servlet 代码，如下所示：

```
Out.print(JspRuntimeLibrary.toString(((num.NumberGuessBean)pageContext.findAttribute("numguess")).getSuccess()));
```

7.2.4 使用示例

下面以一个简单的示例描述如何使用<jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>操作：




```
<jsp:useBean id="checking" scope="session" class="bank.Checking" >
  <jsp:setProperty name="checking" property="balance" value="0.0" />
```

该用户的账号信息为：

```
  <jsp:getProperty name="checking" property="account" />
</jsp:useBean>
```

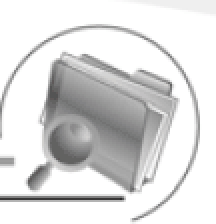
7.3 JSP 与 JavaBean 结合的例子

上面已经对 JavaBean 进行了简单介绍，接下来，来看一些 JSP 与 JavaBean 结合的例子。

7.3.1 计数器 Bean

首先，来看一个简单的计数器程序。本例共包含 2 个文件：Counter.java (JavaBean)文件和 ch7-1.jsp(JSP)文件。Counter.java 主要用于进行计数器的计数操作，ch7-1.jsp 文件用于显示网页的计数。

```
Counter.java
package ch7;
public class Counter
{
  private int count ;
  public Counter()
```



```
    {
        count = 0;
    }
    public int getCount()
    {
        count++;
        return count;
    }
    public void setCount(int value)
    {
        count = value;
    }
}

ch7-1.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<html>
    <head>
        <title>JavaBean 应用示例</title>
    </head>
    <body>
        <jsp:useBean id="bean1" scope="application" class="ch7.Counter" />
        <%
            out.println("当前的计数为： " + bean1.getCount() +"<br>");
        %>
    </body>
</html>
```



Counter.java 创建了一个 Counter 类，其中语句：package ch7 指明了 Counter 类的包文件 ch7。在这个类中，定义了一个构造函数 Counter()和两个方法：getCount()、setCount(int value)。构造函数的作用是在装载 Counter 类时初始化变量 count，将其赋值为 0；setCount(int value)方法是将变量 count 的值改变为参数 value 的值；getCount()方法则是返回变量 count 的值。

再来看看 ch7-1.jsp，从这个例子中，可以看出 JSP 和 JavaBean 应用的一般操作方法：首先在 JSP 页面中要声明并初始化 JavaBean，这个 JavaBean 有一个惟一的 id 标志；还有一个生存范围 scope(设置为 application 是为了实现多个用户共享一个计数器的功能，如果要实现单个用户的计数功能，可以修改 scope 为 session)；最后还要设置 JavaBean 的 class 来源 ch7.counter：

```
<jsp:useBean id="bean1" scope="application" class="ch7.counter" />
```

接着就可以使用 JavaBean 提供的 public 方法 getCount()来得到 JavaBean 中属性 count 的值：

```
out.println("当前的计数为： " + bean0.getCount() + "<BR>");
```

运行程序，如图 7-2 所示。然后多次刷新页面，注意看计数器的变化。

如果要直接在一些 JSP 环境(如 Tomcat、IAS、Weblogic 等)下调试，要注意正确放置 JavaBean 文件。如在 Tomcat 环境中，本例的 JavaBean 编译后的文件需要放在<Server Root>\WEB-INF\classes\ch7\ counter.class 处。

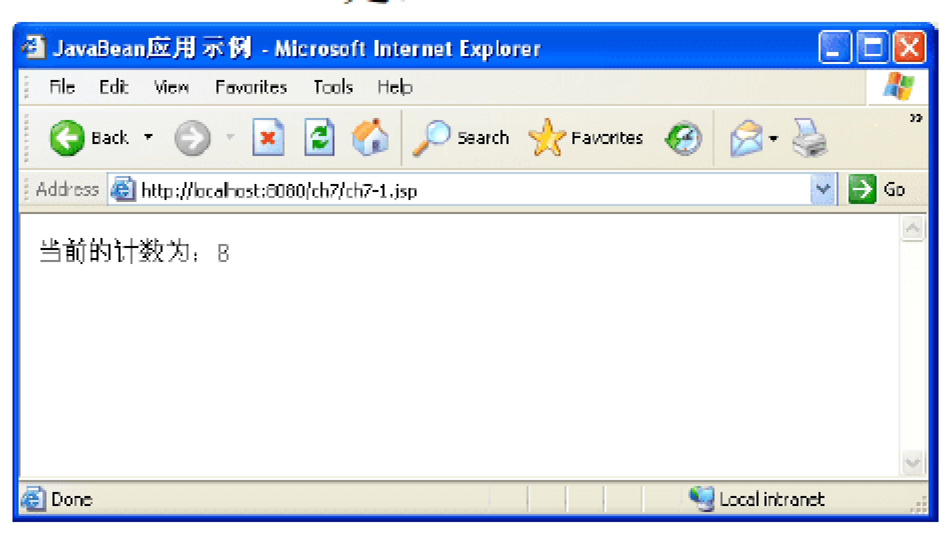


图 7-2　计数器 Bean



7.3.2　税率计算

在上面的例子中已经使用了 JavaBean 中定义的 getCount()方法。但在 JSP 网页中，最常用的是通过<jsp:setProperty>和<jsp:getProperty>操作对 JavaBean 进行访问。下面再给出一个例子，针对直接使用 JavaBean 的方法与使用<jsp:setProperty>和<jsp:getProperty>操作进行具体的说明。下面先看程序代码：

```
//Rate.java
public class Rate {
 private String product;
 private String rate;
 public Rate()
 {
  product = "奶制品";
  rate = "2%";
 }
 public void setProduct(String value)
 {
  product = value;
 }
 public String getProduct()
 {
  return product;
 }
 public void setRate(String value)
```

```
 {
  rate = value;
 }
 public String getRate()
 {
  return rate;
 }
}
```

ch7-2.jsp

```
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<html>
  <head>
  <title>JavaBean 示例二</title>
  </head>
  <body>
    <jsp:useBean id="taxRate" scope="session" class="ch7.Rate" />
    <table border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td>
          <h3>直接调用 JavaBean 的方法：</h3>
          <p>
            修改前：<br>
            产品名称：<%= taxRate.getProduct() %> <br>
            产品的税率：<%= taxRate.getRate() %>
          </p>
          <%
            taxRate.setProduct("蔬菜");
            taxRate.setRate("1%");
          %>
          <p>
            修改后：<br>
            产品名称：<%= taxRate.getProduct() %> <br>
            产品的税率：<%= taxRate.getRate() %>
          </p>
          <h3>调用&ltjsp:getProperty&gt 和&ltjsp:setProperty&gt 操作的方法：</h3>
          <jsp:setProperty name="taxRate" property="product" value="奶制品" />
          <jsp:setProperty name="taxRate" property="rate" value="5%"/>
          <p>
          修改后：<br>
          产品名称：<jsp:getProperty name="taxRate" property="product" /> <br>
```

```
                产品的税率：<jsp:getProperty name="taxRate" property="rate"/>
                </p>
              </td>
            </tr>
          </table>
        </body>
</html>
```

运行程序，结果如图 7-3 所示。

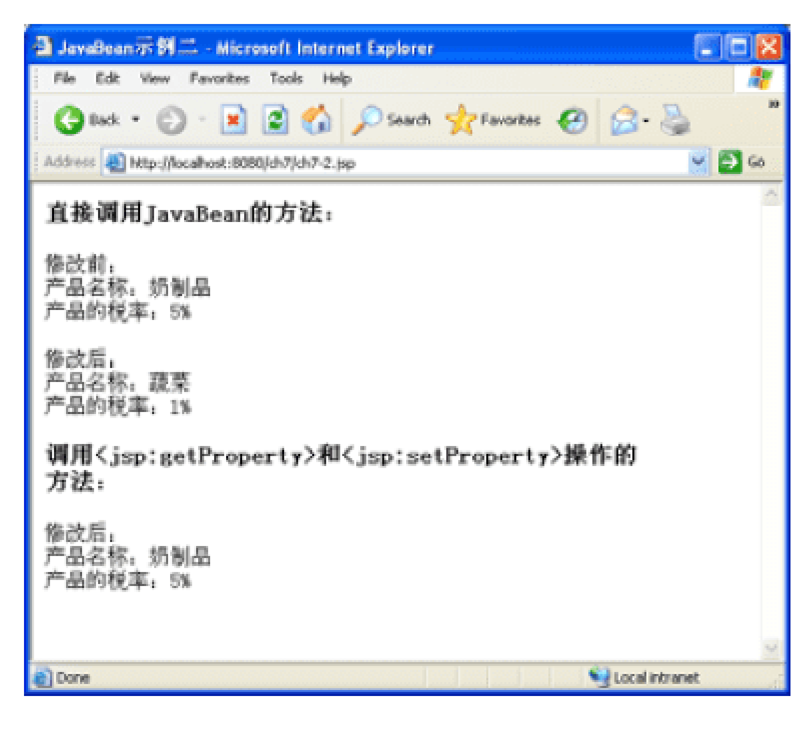
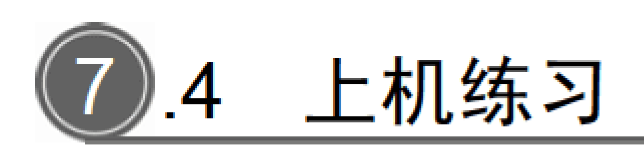


图 7-3　税率计算 Bean



7.4　上机练习

本章上机实验主要练习如何创建 JavaBean 以及如何在 JSP 中使用 JavaBean 进行编程。其中重点掌握如何使用 jsp:useBean、jsp:setProperty 和 jsp:getProperty 等操作。

下面以税率计算为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

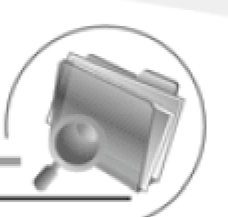
(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮后 Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中将出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Class】命令，弹出 New Java Class 对话框建立一个 JavaBean 类。

(7) 在 New Java Class 对话框的 Package 文本框中输入 Java 类所使用的包名，例如 ch7。

(8) 在 Name 文本框中输入 Java 类的名字，如 Rate.java。



(9) 单击 Finish 按钮即可新建 Java 类。

(10) Eclipse 会创建一个新的类，包括了类的基本框架。在类的编辑窗口中输入 Rate.java 程序代码即可。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 选择【File】|【New】|【JSP】命令，弹出 New JavaServer Page 对话框创建一个 JSP 文件。

(14) 在 New JavaServer Page 对话框的 File name 文本框中输入文件名，如 ch7-2.jsp。

(15) 单击 Finish 按钮，新建 JSP 页面文件。

(16) Eclipse 会创建一个新的 JSP 文件，包括了页面的基本框架。在编辑窗口中输入 ch7-2.jsp 页面代码。

(17) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(18) 根据错误提示，修改所有可能的语法错误。

(19) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(20) 打开 Internet Explorer 浏览器，输入对应的 URL，如：http://localhost:8080/testTomcat/ch7/ch7-2.jsp。观察页面是否正常运行。



7.5 习题

7.5.1 填空题

1. 非可视化 Bean 分为____________和____________两种。
2. 通过实现____________接口可以实现 JavaBean 的持久化。
3. JavaBean 通过____________和____________来读取和设置属性值。

7.5.2 选择题

1. 以下操作中，(　　)是与使用 JavaBean 无关的。

A. jsp:include　　B. jsp:useBean　　C. jsp:setProperty　　D. jsp:getProperty

2. 下面哪个不是 jsp:setProperty 操作的属性(　　)。

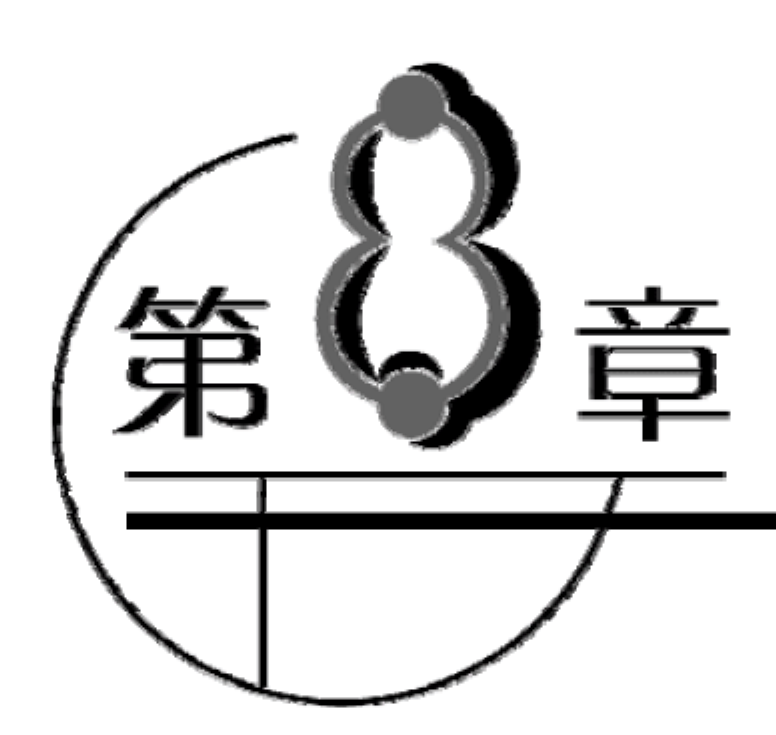
A. name　　B. param　　C. property　　D. scope

7.5.3 问答题

1．Bean 是什么？

2．<jsp:setProperty>和<jsp:getProperty>操作起什么作用？





Servlet 技 术

学习目标

为了更好地理解 JSP，有必要先学习一下它们的底层技术 Java Servlet。Servlet 是通过动态生成 Web 内容而扩展了 Web 服务器功能的 Java 类语言。一个称为 Servlet 引擎的运行环境管理 Servlet 的载入和载出，并结合 Web 服务器将请求导向 Servlet，把输出发回给 Web 客户端。自 1997 年问世以来，Servlet 已经成为服务器端 Java 编程的主要环境，被广泛地使用于应用服务器。

本章重点

- Servlet 的优点
- Servlet 的生命周期
- JSP 与 Servlet 的区别

8.1 Servlet 简介

在学习 Servlet 之前，先来看看什么是 Servlet，它有什么优点。

8.1.1 什么是 Servlet

Servlet 是一种独立于平台和协议的，位于 Web 服务器内部的、服务器端的 Java 应用程序，可以生成动态的 Web 页面。它与传统的从命令行启动的 Java 应用程序不同，Servlet 由 Web 服务器进行加载，该 Web 服务器必须包含支持 Servlet 的 Java 虚拟机。

Java Servlet 与 Applet 有很多相似之处，但二者又有所区别。

- 相似之处：它们都不是独立的应用程序，没有 main()方法；它们都不是由用户或程序员调用，而是由另外一个应用程序(容器)调用；它们都有一个生存周期，包含 init()和 destroy()方法。
- 不同之处：Applet 具有良好的图形界面(AWT)，与浏览器一起，在客户端运行；Servlet 则没有图形界面，在服务器端运行。

Java Servlet 与传统的 CGI 和许多其他类似 CGI 的技术相比，Java Servlet 具有更高的效率、更容易使用、功能更强大，具有更好的可移植性，更能节省投资。

8.1.2 Servlet 的优点

Servlet 的优点主要体现在如下几个方面：

- 可移植性(Portability)

Servlet 是用 Java 语言来开发的，因此延续了 Java 在跨平台上的优势，不论编写 Servlet 的操作系统是 Windows、Solaris、Linux、HP-UX、FreeBSD 或 AIX 等，都能够将写好的 Servlet 程序放在其他操作系统上执行。借助 Servlet 的优势，可以真正达到 Write Once，Serve Anywhere 的境界，这正是 Java 程序员感到最欣慰也是最骄傲的地方。

程序员在开发 Applet 时，经常为了“可移植性”(portability)而感到手忙脚乱，例如：在开发 Applet 时，为了配合 Client 端的平台(即浏览器版本的不同，plug-in 的 JDK 版本也不尽相同)，真正达到“跨平台”的目的，需要花费程序员大量时间来修改程序，以便让所有平台上的用户都能够执行。但即使如此，往往也只能满足大部分用户，而其他少数用户，若要执行 Applet，仍需先安装合适的 JRE(Java Runtime Environment)。

但是 Servlet 就不同了，因为 Servlet 是在 Server 端执行的，所以，程序员只要专心开发，能在实际应用的平台环境下测试无误即可保证所有用户都能执行。

- 强大的功能

Servlet 能够完全发挥 Java API 的优势，包括网络和 URL 存取、多线程(Multi-Thread)、影像处理、RMI(Remote Method Invocation)、分布式服务器组件(Enterprise Java Bean)、对象序列化(Object Serialization)等。如要写个网络目录查询程序，则可利用 JNDI API；想连接数据库，则可以利用 JDBC，有这些强大功能的 API 做后盾，相信 Servlet 更能够发挥其优势。

- 性能

Servlet 在加载执行之后，其对象实体(instance)通常会一直停留在 Server 端的内存中，当有请求(request)发生时，服务器调用 Servlet 来服务，如果收到相同服务的请求，Servlet 会利用不同的线程(thread)来处理，不像 CGI 程序必须产生许多进程(process)来处理数据，在性能方面，大大超越了传统的 CGI 程序。最后补充一点，Servlet 在执行时，不是一直停留在内存中，服务器会自动将停留时间过长且没有执行的 Servlet 从内存中清除，不过有时也可以自行写程序来控制。至于停留时间的长短通常与服务器的类型有关。



◉ 安全性

Servlet 也有类型检查(Type Checking)的特性，并且利用 Java 的垃圾收集(Garbage Collection)与没有指针的设计，这些都可以使得 Servlet 避免了内存管理的问题。

在 Java 的异常处理(Exception-Handling)机制下，Servlet 能够安全地处理各种错误，因此，也就不会因为发生程序上的逻辑错误而导致服务器系统的毁灭。例如：某个 Servlet 发生除以零或其他不合法的运算时，它会抛出一个异常(Exception)让服务器处理，如记录在记录文件中(log file)。

8.2 Servlet 的应用

本节将具体介绍 Servlet 的应用，包括 Servlet 的基本结构、编译以及生命周期等。

8.2.1 Servlet 的基本结构



下面是一个处理 GET 请求的 Servlet。GET 请求是浏览器对网页的基本请求。当用户在地址栏中输入一个 URL，单击网页上的超链接或者提交一个没有指定 METHOD 的 HTML 表单时浏览器都会发送这个请求。Servlet 也可以处理 POST 请求，POST 请求在提交一个指明 METHOD="POST"的 HTML 表单时发出。一个 Servlet 类应该继承 HttpServlet 并且覆盖 doGet 或者 doPost 方法，这取决于数据的发送方式是 GET 还是 POST。如果要在一个 Servlet 中同时处理 GET 和 POST 请求，并且对每个请求的处理动作相同，可以简单地在 doGet 方法中调用 doPost 方法，或者反过来在 dopost 方法中调用 doGet 方法。

代码清单：

```
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
public class ServletTemplate extends HttpServlet
{
    public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException
    {
      // 用"request" 可以读取输入的值(例如：cookies)
      // 用"response" 返回输出值
      PrintWriter out = response.getWriter();
      // 用"out" 向浏览器写内容
    }
}
```

doPost 和 doGet 方法都有两个参数：HttpServletRequest 和 HttpServletResponse。可以通过 HttpServletRequest 类提供的方法获得引入的信息，如表单数据、HTTPrequestheader、

客户主机名等。HttpServletResponse 类提供输出信息的能力：比如可以指定 response headers(Content-Type、Set-Cookie 等)，最重要的是可以通过这个参数得到一个 PrintWriter 向客户发送文档内容。作为一个简单的 Servlet，最主要的功能就是用 println 语句输出一个预期的页面。因为 doGet 和 doPost 会抛出两个异常，所以需要引入相关的类：java.io(因为要用到 PrintWriter 等)、javax.Servlet(因为要用到 HttpServlet 等)以及 javax.Servlet.http(因为要用到 HttpServletRequest 和 HttpServletResponse 等)。

8.2.2 Servlet 的编译、配置和调用

1. Servlet 的编译和配置

首先要确定服务器是否已经正确配置，环境变量 CLASSPATH 是否包含了标准的 Servlet 类，可以参考本书的第 2 章来进行这些工作。当配置好服务器并设定了 CLASSPATH 变量之后，将 Servlet 放置在正确的目录，然后用“javac”命令编译 Servlet。将生成的 Servlet 类放到正确的路径：ROOT\WEB-INF\classes\。

完成上面的步骤后，Servlet 必须使用\ROOT\WEB-INF 目录下的 web.xml 文件进行注册。打开这个 web.xml 文件，在里面加入如下模块(假设创建的 Servlet 类为 HelloWorld.class)：

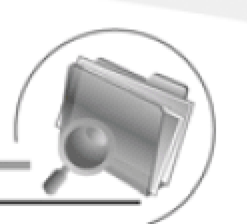
```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/servlet/helloworld</url-pattern>
</servlet-mapping>
```

结构：

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>
```

表示指定包含的 Servlet 类。servlet-name 命名该 Servlet 类，而 servlet-class 则指出该类的类名(class 文件)。

而下面的结构：



```
<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/servlet/HelloWorld</url-pattern>
</servlet-mapping>
```

表示指定 Servlet 应当映射到哪一个 URL 模式。

在修改完 web.xml 之后，重新启动 Tomcat 服务。

2. Sevlet 的调用

在本书中，应用程序的目录为根目录，所以调用 Servlet 的 URL 为 http://localhost/servlet/ServletName。

8.2.3　Servlet 的生命周期



当 Servlet 第一次被创建时，init 方法会被调用，所以一次性设置的代码应该存在于 init 方法中。用户的请求会导致创建新的线程来调用之前创建的 Servlet 实例的 service 方法。并发的多个请求会创建调用 service 方法的多个线程。也可以让 Servlet 继承一个特殊的接口来保证在任何时间只能有一个线程在运行。Service 方法会根据收到的 HTTP 请求的不同调用不同的 doXxx 方法(doGet、doPost 等)。最后，服务器会调用 destroy 方法来卸载 Servlet。

1. init 方法

init 方法只有在 Servlet 首次被创建时调用，收到用户请求时并不调用 init 方法。所以，它常被用来作为一次性初始化的工作，这点类似于 Applet 中的 init 方法。Servlet 会在用户首次调用一个 Servlet 的 URL 或服务器启动之时被创建，这取决于是否在 Web 服务器中注册了 Servlet。当收到第一个用户对这个 Servlet 的请求且该 Servlet 只是正确存在于服务器的特定路径而并未明确注册时，Servlet 会被创建。

init 方法有两个版本：一个是没有参数；另一个以 ServletConfig 对象作为参数。在 Servlet 初始化时，如果不需要任何服务器的设置信息，则使用无参数的 init 方法。无参数的 init 方法定义如下：

```
public void init() throws ServletException
{
   // 初始化代码…
}
```

当 servlet 需要获得服务器的设置信息才能完成初始化时，就要使用 init 的第二个版本。举例来说，servlet 可能需要有关数据库的设置、password 文件、服务器特定性能参数单击计数文件、之前请求包含的 serialized cookie 数据等信息。带参数的 init 方法定义如下：

```
public void init(ServletConfig config) throw ServletException
{
    super.init(config);
    //初始化代码…..
}
```

这段代码有两个地方需要注意：第一，这里的 init 方法有一个 ServletConfig 对象作为参数。ServletConfig 类提供一个 getInitParameter 方法以便用户可以获取与 Servlet 关联的初始化参数。第二，方法体的第一行调用了 super.init，这非常重要，ServletConfig 对象会在 Servlet 的其他地方用到，超类的 init 方法会将 ServletConfig 注册以便 Servlet 之后用到。



提示

如果使用以 ServletConfig 为参数的 init 方法，那么，一定要在方法体的第一行调用 super.init()。

2. service 方法

当服务器接收到对 Servlet 的请求时，服务器会产生一个新的线程调用 service 方法。service 方法首先检查 HTTP 请求类型(GET、POST、PUT、DELETE 等)，然后调用相应的 doGet、doPost、doPut 或 doDelete 方法。如果 Servlet 处理 POST 请求和 GET 请求的方式相同，也可以尝试覆盖 service 方法(如下所示)或者同时实现 doGet 和 doPost 方法。

```
public void service(HttpServletRequest request,
HttpServletResponse    response)
throws ServletException, IOException
{
  // Servlet 代码 }
}
```

采用这种方式并不理想，更好的做法是在 doPost 方法中调用 doGet 方法(或者反过来)，如下所示：

```
public void doGet(HttpServletRequest   request,
HttpServletResponse   response)
throws ServletException, IOException
{
  // Servlet 代码
}
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
```

```
{
  doGet(request, response);
}
```

虽然在第 2 种方法中代码稍多了一些，但相比直接覆盖 service 方法它有 5 个优点：第一，这样做确保以后可以在子类中添加对其他请求服务的支持，如 doPut、doTrace 等，如果直接覆盖了 service 方法就没有这种可能性了。第二，可以通过实现 getLastModified 方法来增加对修改日期的支持，如果调用了 doGet 方法，标准 service 方法会用 getLastModified 方法设置 header 的最后修改日期，然后对 GET 请求作出相应的回应(包括已修改过的 header)。第三，可以自动获得对 HEAD 请求的支持，系统将只返回 doGet 方法设定的 header 和 status code，而忽略页面体，对于定制 HTTP 客户来说 HEAD 是个很有用的请求方法。例如检查一个页面中的死链时通常使用 HEAD 替代 GET 以减轻服务器负担。第四，可以自动获得对 OPTIONS 请求的支持，如果 doGet 方法退出，标准 service 方法通过返回一个 Allow header 来询问 OPTION，Allow header 指示对 GET、HEAD、OPTION 和 TRACE 的支持。第五，可以自动获得对 TRACE 请求的支持，TRACE 请求方法用于客户端的调试，它只返回 HTTP 请求 header 给客户端。


提示

如果 Servlet 处理 GET 和 POST 方式相同，那么可以让 doPost 方法调用 doGet 方法(或者反过来)，而不要直接覆盖 service 方法。

doGET、doPost 及其他 doXxx 方法是 Servlet 的主体，大部分时间都只用关心 GET 和 POST 请求，所以需要覆盖 doGet 和 doPost 方法。如果需要，也可以为了处理 DELETE 请求而覆盖 doDelete 方法；为处理 PUT 请求而覆盖 doPut 方法；为处理 OPTIONS 请求覆盖 doOptions 方法；为处理 TRACE 请求覆盖 doTrace 方法。会自动获得对 OPTIONS 和 TRACE 的支持，如前文所提到的一样。注意这里没有 doHead 方法。因为系统自动用状态列(status line)和 header 设置来回复 HEAD 请求。

下面简单介绍 SingleThreadModel 接口。

一般情况下，系统为每个 Servlet 创建一个实例，为每个请求创建一个线程。当一个新的请求到来而另一个请求产生的线程还在执行时，就会有多个线程并发执行。这就意味着 doGet 和 doPost 方法必须注意到共享数据和领域的同步访问问题，因为多个线程可能会同时尝试访问同一块数据。如果要避免多线程的并发访问，就可以使 Servlet 实现 SingleThreadModel 接口，如下所示：

```
public class YourServlet extends HttpServlet
implements SingleThreadModel
{
    ...
}
```

如果实现了这个接口，系统就会确保 Servlet 的一个实例同时只会被一个请求线程访问。这

通过两种方法实现：其一，系统将请求放入队列中，在同一时间只传递一个给 Servlet 的单个实例；其二，创建一个多实例池，每个实例同一时间只控制一个请求，这就使用户不必担心 Servlet 的实例变量的并发访问。但是，仍然需要同步化类变量(静态变量)和 Servlet 之外的共享数据的访问。

3. destroy 方法

如果服务器管理员明确要求卸载一个 Servelt 实例，或者某 Servlet 已经空闲了很长时间，服务器会移除先前装载的 Servlet 实例。移除之前服务器会调用 Servlet 的 destroy 方法。Servlet 可以使用这个方法关闭数据库连接、中断后台线程、向磁盘写入 cookie 列表、保存点击数等数据以及其他清理动作。

8.2.4 Servlet 类

这一节将概括介绍 javax.servlet 和 javax.servlet.http 包中的几个重要类。

1. Servlet

javax.servlet.servlet 公共接口：该接口定义了必须由 Servlet 类实现，由 Servlet 引擎识别和管理的方法集，如表 8-1 所示。

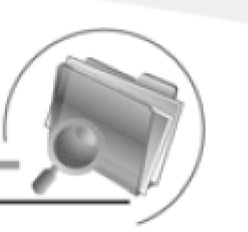
表 8-1　Servlet 接口中的方法

方　法	描　述
void destroy()	当 Servlet 将要卸载时由 Servlet 容器调用
ServletConfig getServletConfig()	返回 ServletConfig 对象，该对象包括该 Servlet 的初始化和启动信息
String getServletInfo()	返回该 Servlet 的相关信息，例如版本、版权等
void init(ServletConfig config)	在 Servlet 被载入后，并且在实施服务前由 Servlet 容器进行
void service(ServletRequest req, ServletResponse res)	处理来自 request 对象的请求，并使用 response 对象返回请求结果

Servlet API 提供了 Servlet 接口的直接实现，称为 GenericServlet，参看表 8-2 所示。GenericServlet 类提供了除 service()方法之外的所有接口中方法的默认实现。这表示通过简单扩展 GenericServlet 类和编写一个定制的 service()方法就可以编写一个基本的 Servlet。

表 8-2　GenericServlet 类中的方法

方　法	描　述
void destroy()	当 Servlet 将要卸载时由 Servlet 容器调用
String getInitParameter(String name)	返回命名的初始化参数值，参数不存在则返回 null
Enumeration getInitParameterNames()	返回初始化参数的名字集，如果没有参数则返回 null
ServletConfig getServletConfig()	返回该 Servlet 的 ServletConfig 对象
ServletContext getServletContext()	获得该 Servlet 的 ServletContext 引用



(续表)

方　法	描　述
String getServletInfo()	返回该 Servlet 的相关信息，例如版本、版权等
String getServletName	返回该 Servlet 实例的名字
void init()	在 Servlet 初始化时不需要任何服务器的设置信息时使用该无参数的 init 方法
void init(ServletConfig config)	当 Servlet 需要获得服务器的设置信息才能完成初始化时使用该带有参数的 init 方法
void log(String msg)	将特定的信息写入到 Servlet 的日志文件中
abstract void service(ServletRequest req, ServletResponse res)	处理来自 request 对象的请求，并使用 response 对象返回请求结果

Servlet 一般创建方法的是扩展其指定的 HTTP 子类 HTTPServlet，参看表 8-3 所示。HTTPServlet 通过调用指定到 HTTP 请求的方法实现 service()，亦即对于 DELETE、HEAD、OPTIONS、GET、POST、PUT 和 TRACE，分别调用 doDelete()、doHead()、doOptions()、doGet()、doOptions()、doPost()、doPut()和 doTrace()方法。



表 8-3　HTTPServlet 类中的方法

方　法	描　述
void doDelete(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP DELETE 请求
void doGet(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP GET 请求
void doHead(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP GET 请求
void doOptions(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP OPTIONS 请求
void doPost(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP GET 请求
void doPut(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP PUT 请求
void doTrace(HttpServletRequest req, HttpServletResponse rep)	调用处理一个 HTTP Trace 请求
Long getLastModified(HttpServletRequest req)	返回 HttpServletRequest 对象被修改的最新时间
void service(HttpServletRequest req, HttpServletResponse rep)	接收 HTTP 请求并把请求分派到相应的 doXXX 方法

下面，看一个简单的实例：在程序清单 HTMLPage.java 中，将生成一个完整的 HTML 页面用于显示一个简单的文本字符串。这个 Servlet 扩展了 HttpServlet 类并且重载了 doGet()方法。

```
HTMLPage.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLPage extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse res){
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<html>");
   out.println("<head><title>第一个 Servlet！ </title></head>");
   out.println("<body>");
   out.println("<h1>第一个由 Servlet 直接产生的 HTML 网页！ </h1>");
   out.println("</body>");
   out.println("</html>");
 }
}
```

按照 8.2.2 节的方法编译并配置 Servlet 后，调用该 Servlet，运行结果如图 8-1 所示。



图 8-1 一个简单的 Servlet 示例

2. Servlet 请求

ServletRequest 接口封装了客户端请求的细节，HTTPServletRequest 是 ServletRequest 接口的子接口，HTTPServletRequest 接口的方法参看表 8-4。

表 8-4 HTTPServletRequest 接口方法

方　法	描　述
String getAuthType()	返回保护该 Servlet 鉴定方案的名称
Strig getContextPath()	返回指定 Servlet 上下文的 URI 前缀
Cookie[] getCookies()	返回与请求相关的 cookie 对象数组
long getDateHeader(String name)	返回指定的请求头域的值，该值被转换成一个自 1970-1-1 日(GMT)以来精确到毫秒的长整数



(续表)

方　　法	描　　述
String getHeader(String name)	将指定请求头的值作为 String 类型返回
Enumeration GetHeaderNames()	返回请求给出的所有 HTTP 头名称的枚举类型值
int getIntHead(String name)	返回指定的请求头域的值，该值被转换成一个整数
String getMethod()	返回这个请求使用的 HTTP 方法(例如：GET、POST、PUT)
String getPathInfo()	返回在请求 URL 信息中位于 Servlet 路径之后的额外路径信息。如果这个 URL 包括一个查询字符串，在返回值内将不包括这个查询字符串。这个路径在返回之前必须经过 URL 解码。如果在请求的 URL 的 Servlet 路径之后没有路径信息，该方法返回空值
String getQueryString()	返回这个请求 URL 所包含的查询字符串
String getRemoteUser()	返回作了请求的用户名，这个信息用来作 HTTP 用户论证
String getRequestdSessionId()	返回这个请求相应的 session id。如果由于某种原因客户端提供的 session id 是无效的，该 session id 将与在当前 session 中的 session id 不同，与此同时，将建立一个新的 session
String getRequestURI()	从 HTTP 请求的第一行返回请求的 URL 中定义被请求的资源部分
String getServletPath()	返回请求 URL 反映调用 Servlet 的部分
HttpSession getSession()	返回与这个请求关联的当前有效的 session。如果调用这个方法时没带参数，那么在没有 session 与这个请求关联的情况下，将会新建一个 session。如果调用这个方法时带入了一个布尔型的参数，只有当这个参数为真时，session 才会被建立
bool isRequestedSessionId Valid()	这个方法检查与此请求关联的 session 当前是不是有效。如果当前请求中使用的 session 无效，它将不能通过 getSession 方法返回
bool isRequestedSessionId FromCookie()	如果这个请求的 session id 是通过客户端的一个 cookie 提供的，该方法返回真，否则返回假
bool isRequestedSessionId FromURL	如果这个请求的 session id 是通过客户端的 URL 的一部分提供的，该方法返回真，否则返回假



下面来看一个简单的实例：

```
RequestInfo.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
public class RequestInfo extends HttpServlet {
  ResourceBundle rb = ResourceBundle.getBundle("LocalStrings");
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException{
          res.setContentType("text/html;charset=gb2312");
```

```
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>请求信息示例</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"skyblue\">");
        out.println("<h3 align=center>请求信息示例</h3>");
        out.println("<center><table border=1><tr><td>方法</td><td>");
        out.println(req.getMethod());
        out.println("</td></tr><tr><td>请求 URI</td><td>");
        out.println(req.getRequestURI());
        out.println("</td></tr><tr><td>协议</td><td>");
        out.println(req.getProtocol());
        out.println("</td></tr><tr><td>路径信息</td><td>");
        out.println(req.getPathInfo());
        out.println("</td></tr><tr><td>远程地址</td><td>");
        out.println(req.getRemoteAddr());
        out.println("</td></tr>");
       String cipherSuite=
                    (String)req.getAttribute("javax.servlet.request.cipher_suite");
        if(cipherSuite!=null){
        out.println("<tr><td>");
        out.println("SSLCipherSuite:");
        out.println("</td>");
        out.println("<td>");
        out.println(req.getAttribute("javax.servlet.request.cipher_suite"));
          out.println("</td>");
        }
        out.println("</table></center>");
        out.println("</body>");
        out.println("</html>");
 }
    public void doPost(HttpServletRequest req,
                 HttpServletResponse res)
     throws IOException, ServletException
    {
        doGet(req, res);
    }
}
```

调用该 Servlet，运行结果如图 8-2 所示。



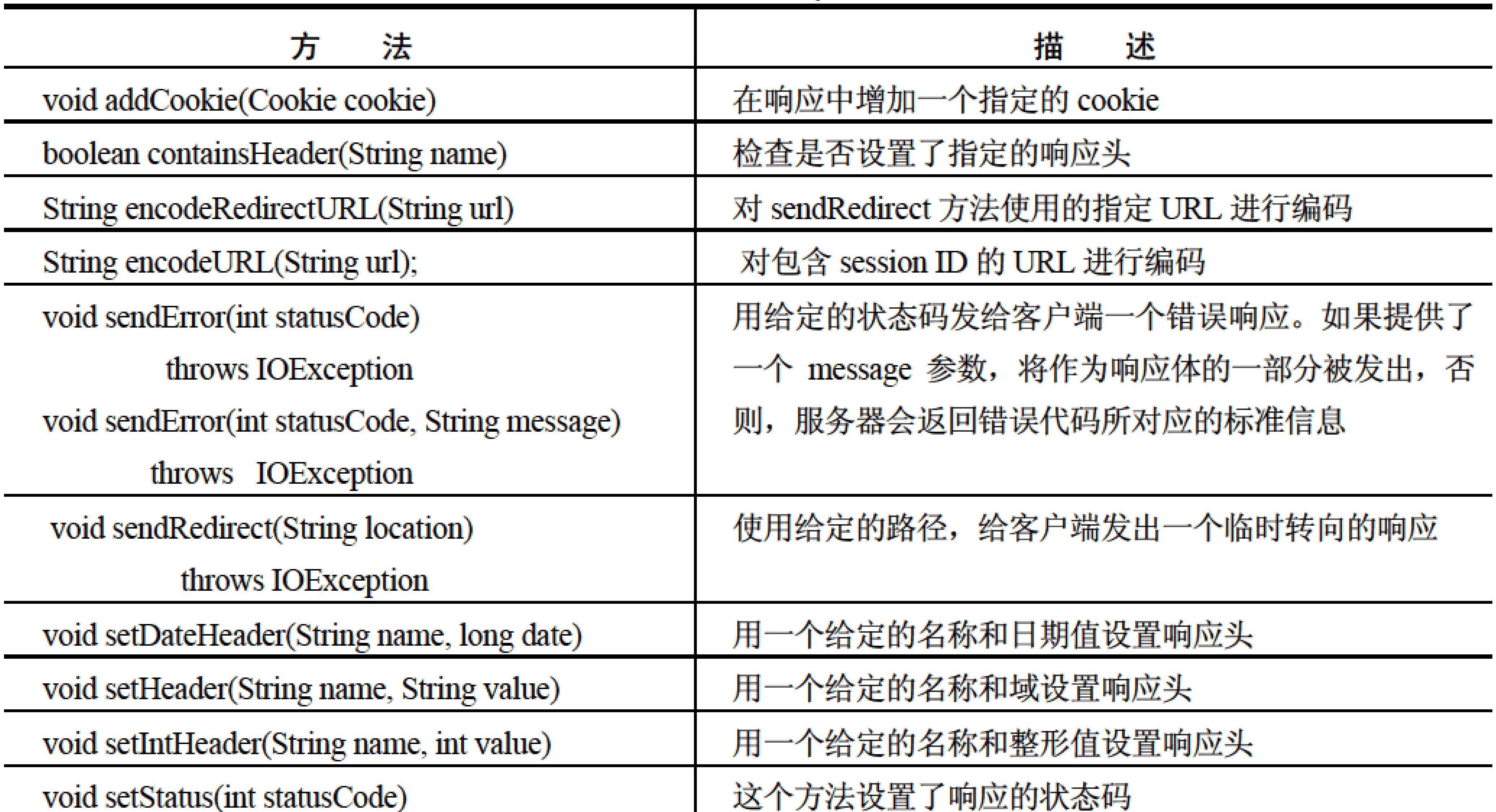


图 8-2　HTTPServletRequest 接口方法演示

3. Servlet 响应

ServletResponse 接口封装了服务器响应客户请求的细节，HTTPServletResponse 接口是 ServletResponse 接口的子接口，HTTPServletResponse 接口的方法参看表 8-5 所示。


表 8-5　HTTPServletResponse 接口方法



方　法	描　述
void addCookie(Cookie cookie)	在响应中增加一个指定的 cookie
boolean containsHeader(String name)	检查是否设置了指定的响应头
String encodeRedirectURL(String url)	对 sendRedirect 方法使用的指定 URL 进行编码
String encodeURL(String url);	对包含 session ID 的 URL 进行编码
void sendError(int statusCode) throws IOException void sendError(int statusCode, String message) throws IOException	用给定的状态码发给客户端一个错误响应。如果提供了一个 message 参数，将作为响应体的一部分被发出，否则，服务器会返回错误代码所对应的标准信息
void sendRedirect(String location) throws IOException	使用给定的路径，给客户端发出一个临时转向的响应
void setDateHeader(String name, long date)	用一个给定的名称和日期值设置响应头
void setHeader(String name, String value)	用一个给定的名称和域设置响应头
void setIntHeader(String name, int value)	用一个给定的名称和整形值设置响应头
void setStatus(int statusCode)	这个方法设置了响应的状态码

Servlet 相关类和接口中的各种方法，读者可以在自己编写代码的过程中进一步学习和理解。

4. Servlet 获取表单参数的实例

Java Servlet 对表单的处理功能由 HTML 表单和 Java Servlet 程序完成。首先完成 HTML 表单提交程序，表单提交以后，表单中的一些上传参数可以在服务器端的 Servlet 程序中进行处理，如填入数据库、查询、网上购物等，下面是这 2 个文件的详细代码。



```
//FormGet.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<html>
  <head>
    <title>表单传递参数</title>
  </head>
  <body>
    <h3 align="center">请输入用户信息</h3>
    <form method="get"
          action="http://localhost:8080/servlet/GetFormParameter">
      <table border="1" cellspacing="1" align="center">
        <tr>
          <td width="100" height="10">姓名：</td>
          <td width="300"> <input type="text" name="username"> </td>
        </tr>
        <tr>
          <td width="100" height="10">密码:</td>
          <td width="300"> <input type="text" name="password"> </td>
        </tr>
        <tr>
          <td width="100" height="10">确认密码:</td>
          <td width="300"> <input type="text" name="confirmpwd"> </td>
        </tr>
        <tr>
          <td width="100" height="10">性别:</td>
          <td width="300">
            <input type="radio" name="gender" value="1">男
            <input type="radio" name="gender" value="2">女
          </td>
        </tr>
        <tr>
          <td width="100" height="10">区域:</td>
          <td width="300">
            <select name="area" >
              <option value="1" selected="true">北京</option>
              <option value="2">上海</option>
              <option value="3">成都</option>
              <option value="4">广州</option>
              <option value="5">深圳</option>
            </select>
```

```
            </td>
          </tr>
        </table>
        <p align="center">
          <input type="submit" name="submit" value="提　交" width="50">
          <input type="reset" name="reset" value="重　置">
      </form>
    </body>
</html>
```

FormGet.jsp 的作用是生成一个表单，在浏览器中显示如图 8-3 所示。



图 8-3　FormGet.jsp 显示结果



文件 GetFormParameter.java：

```
import javax.servlet.http.*;
import java.io.*;

public class GetFormParameter extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws IOException{
    res.setContentType("text/html;charset=gb2312");
    PrintWriter out = res.getWriter();
    out.println("<html>");
    out.println("<head><title>用 servlet 获取表单参数</title></head>");
    out.println("<body>");
    out.println("<h1>用 servlet 获取表单参数示例</h1>");

    String name = new String(req.getParameter("username").getBytes("ISO8859_1"));
    out.println("<h4>用户姓名：" + name + "</h4>");
```

```
        String pwd = req.getParameter("password");
        out.println("<h4>用户密码：" + pwd + "</h4>");
        String gender = req.getParameter("gender");
        String value="";
        if(gender.equals("1"))
          value = "男";
        if(gender.equals("2"))
          value = "女";
        out.println("<h4>用户性别：" + value + "</h4>");
        String area = req.getParameter("area");
        value="";
        if(area.equals("1"))
          value = "北京";
        else if(area.equals("2"))
          value = "上海";
        else if(area.equals("3"))
          value = "成都";
        else if(area.equals("4"))
          value = "广州";
        else if(area.equals("5"))
          value = "深圳";
        out.println("<h4>用户所在的区域是：" + value + "</h4>");

        out.println("</body>");
        out.println("</html>");
    }
}
```

用浏览器打开 FormGet.jsp，输入用户信息后，单击【提交】按钮就可以运行 GetFormParameter Servlet 程序了，运行结果如图 8-4 和图 8-5 所示。



图 8-4　表单信息



图 8-5　GetFormParameter servlet 运行结果



8.3　JSP 和 Servlet

本书将介绍 JSP 和 Servlet 的区别，以及如何让两者结合起来使用。

8.3.1　JSP 与 Servlet 的区别

JSP 是一种脚本语言，简化了 Java 和 Servlet 的使用难度，同时通过扩展 JSP 标签(TAG)，提供了网页动态执行的能力。尽管如此，JSP 仍然没有超出 Java 和 Servlet 的范围，JSP 页面上可以直接编写 Java 代码，JSP 是先被编译成 Servlet 之后才实际运行的。JSP 在服务器上执行，并将执行结果输出到客户端浏览器，可以说基本上与浏览器无关。它与 JavaScript 不同，JavaScript 是客户端脚本语言，在客户端解释并执行，与服务器无关。

JSP 与 Servlet 之间的主要差异在于：JSP 提供了一套简单的标签，和 HTML 融合得比较好，可以使不了解 Servlet 的人也能做出动态网页来。使不熟悉 Java 语言的人也会觉得 JSP 开发比较方便。JSP 修改后可以立即看到结果，不需要手工编译，JSP 引擎会完成这些工作；而 Servelt 需要编译，重新启动 Servlet 引擎等一系列动作。但是在 JSP 中，HTML 与程序代码混杂在一起，而 Servlet 却不是这样。



下面对 JSP 的运行来做一个简单的介绍，告诉大家怎样来执行一个 JSP 文件：当 Web 服务器(或 Servlet 引擎、应用服务器)支持 JSP 引擎时，JSP 引擎会按照 JSP 的语法，将 JSP 文件转换成 Servlet 代码源文件，接着 Servlet 会被编译成 Java 可以执行的字节码，并以一般的 Servlet 方式载入执行。

JSP 语法简单，可以方便地加入 HTML 之中，很容易加入动态部分，方便地输出 HTML。在 Servlet 中输出 HTML 却需要调用特定的方法，对于引号之类的字符也要做特殊的处理，相对 JSP 来说是要困难一些。

除了转换和编译阶段，JSP 和 Servlet 之间的区别则不大。

8.3.2　选择 JSP 还是 Servlet

自从引入 JSP 技术之后，使用 Java 构建的 Web 应用服务器端出现了两种架构：第一种是只使用 JSP；第二种是同时使用 JSP 和 Servlet，下面分别称之为模型一和模型二，它们各有各的优缺点。选择模型一还是模型二完全取决于目前项目的实际需求，下面简单描述这两种模型，权衡利弊，提供一些基本的准则来帮助用户决定哪一种技术更适合。

通常情况下，使用 JSP 和 JavaBean、标记库相结合。用 JSP 开发 Web 应用程序最大的优

点就是开发速度快，页面风格和功能转变迅速。

如果 Java 程序段非常复杂，则应该考虑使用模型二：同时使用 JSP 和 Servlet。支持复杂 Java 程序段的常用方式是用 Servlet 进行复杂的逻辑处理，用 JSP 处理页面的表现形式。Servlet 把数据放到会话环境中，提交 HTTP 请求给 JSP，JSP 从会话环境中取出数据，组织成 HTML 页面输出到客户端，这种方法支持表达层和逻辑层的分离。Servlet 和 JSP 结合带来的另一个好处是，Servlet 根据发出请求的客户端类型调用不同的 JSP，满足不同类型的显示需求。例如：可以为 HTML 客户端(PC 机)定义一个 JSP，为 WML 客户端(手机)定义另一个 JSP，还有别的一些客户端等。所有的这些客户端请求都由单个的 Servlet 来处理，当处理逻辑发生变化时，只需要修改一下 Servlet 和相应的 JSP 显示部分即可。

8.4 上机练习

本章上机实验主要练习如何创建和配置 Servlet，以及如何将 JSP 和 Servlet 结合起来使用。

下面以用 Servlet 获取表单参数为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮后 Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中将出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Servlet】命令，弹出 Create Servlet 对话框建立 Servlet 类。

(7) 在 Create Servlet 对话框的 Java Package 文本框中输入 Servlet 类所使用的包名，或者为空。

(8) 在 Class Name 文本框中输入 Servlet 类的名字，如 GetFormParameter.java。

(9) 单击 Finish 按钮完成 Servlet 类的创建。

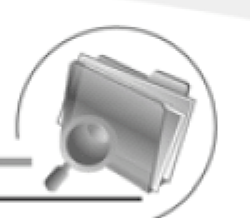
(10) Eclipse 会创建一个新的 Servlet 类，包括了基本的 Servlet 框架。在类的编辑窗口中输入 GetFormParameter.java 程序代码。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 选择【Project】|【Build All】命令编译 GetFormParameter.java Servlet 文件。

(14) 检查工作空间 testTomcat 项目 WEB-INF 目录下的 classes 文件夹，可以找到相应的



GetFormParameter.class 文件。

(15) 修改 WEB-INF 下的 web.xml 文件，加入<Servlet>和<Servlet-mapping>配置模块进行 Servlet 类的配置。

(16) 选择【File】|【New】|【JSP】命令弹出 New JavaServer Page 对话框创建一个 JSP 文件。

(17) 在 New JavaServer Page 对话框的 File name 文本框中输入文件名，如 FormGet.jsp。

(18) 单击 Finish 按钮新建 JSP 页面文件。

(19) Eclipse 会创建一个新的 JSP 文件，包括了基本的页面框架。在编辑窗口中输入 FormGet.jsp 页面代码。检查页面代码中相应的 Servlet URL 是否正确。

(20) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(21) 根据错误提示，修改所有可能的语法错误。

(22) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(23) 打开 Internet Explorer 浏览器，输入相应的 URL，如 http://localhost:8080/ch8/FormGet.jsp。观察页面是否正常运行。

8.5 习题

8.5.1 填空题

1. Servlet 的优点有____________、____________、____________和____________。
2. 如果使用以 ServletConfig 为参数的 init 方法，那么一定要在方法体第一行调用________。

8.5.1 选择题

1. 以下方法中，哪一个方法不是 HTTPServlet 类的方法(　　)。

A. doGet　　B. doService　　C. doPost　　D. doDelete

2. 以下类中，哪一个类不是 javax.servlet 和 javax.servlet.http 包中提供的类或接口(　　)。

A. Servlet　　B. BaseServlet　　C. GenericServlet　　D. HTTPServlet

8.5.3　问答题

1. 必须实现以处理 GET 和 POST 请求的方法的名字是什么？
2. JSP 和 Servlet 的区别是什么？



JSP 标记库

学习目标

JSP 引入了十分简单而实用的新功能：自定义 JSP 标记。开发人员可以定义标记和属性及其内容是怎么被解析，然后将所有的标记组成一个集合，称为标记库。任意的 JSP 文件都可以使用这个标记库。这种自定义标记库的能力使得 Java 开发人员可以将复杂服务器端行为简化为简单而容易的元素，JSP 开发人员也就更容易地将之融入到他们的 JSP 页面中。自定义标记和使用 jsp:useBean 来访问的 Beans 都可以完成同样的功能：将复杂的行为封装为简单而可访问的形式。JSP 标记库是对 JSP 标准标记的扩展，标记库由用户自己定义，标记的功能由 Java 程序实现并应用于 JSP 的处理过程中。

本章重点

- 标记库描述符文件
- 标记处理类
- 自定义标记的生命周期
- 定义脚本变量的标记

9.1 什么是自定义标记

一个自定义的 tag 标签是指用户定义的一种 JSP 标记。当一个含有自定义 tag 标签的 JSP 页面被 JSP 引擎编译成 Servlet 时，tag 标签将被转化成了对 tag 处理类的对象进行的操作。于是，当 JSP 页面被 JSP 引擎转化为 Servlet 之后，实际上 tag 标签被转化成为了对 tag 处理类的操作。

自定义 tag 标签有很多特色，诸如：可以在 JSP 页面中自定义 tag 标签的属性；访问 JSP 页面中的所有对象；可以动态地修改页面输出；彼此之间可以相互通信。可以先创建一个

JavaBeans 组件，然后在一个 tag 中调用此 JavaBeans 组件，同时可以在另一个 tag 中调用它；tag 允许相互嵌套，可以在一个 JSP 页面中完成一些复杂的交互。

9.2 开发简单的自定义标记

一个自定义标记由 Java 类文件和标记库描述符两部分组成。Java 类文件，实现标记功能；标记库描述符(TLD)，定义标记名、实现标记的 Java 类以及有关部署和使用标记的信息。

使用自定义标记，必须在 JSP 的开始处引用 TLD 文件，在一个 JSP 页面中可以使用多个标记库。在引入 TLD 文件之后，可以像使用普通的 JSP 标记一样使用 TLD 文件定义的标记。

9.2.1 使用简单的标记

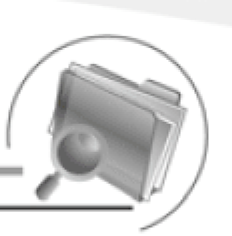


开始学习编写和使用标记库之前，先来实现一个简单的自定义标记。这个例子虽然不能体现自定义标记的优点，但可以先了解一下自定义标记的使用规则和部署过程。

程序清单 ShowServer.jsp 演示了在 JSP 页面中显示服务器名字的简单标记。

```
//ShowServer.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<%@ taglib prefix="demo" uri="/WEB-INF/test.tld"%>
<html>
  <head>
    <title>自定义标记演示</title>
  </head>
  <body>
  <center>
    <h3>简单的自定义标记演示</h3>
    服务器名字是：
   <demo:getWebServer/>
  </center>
  </body>
</html>
```

程序在一开始就定义了一个标记库，并把前缀 demo 与标记库里的标记联系起来。在第 12 行使用了一个名为 getWebServer 的标记，输出的页面如图 9-1 所示。



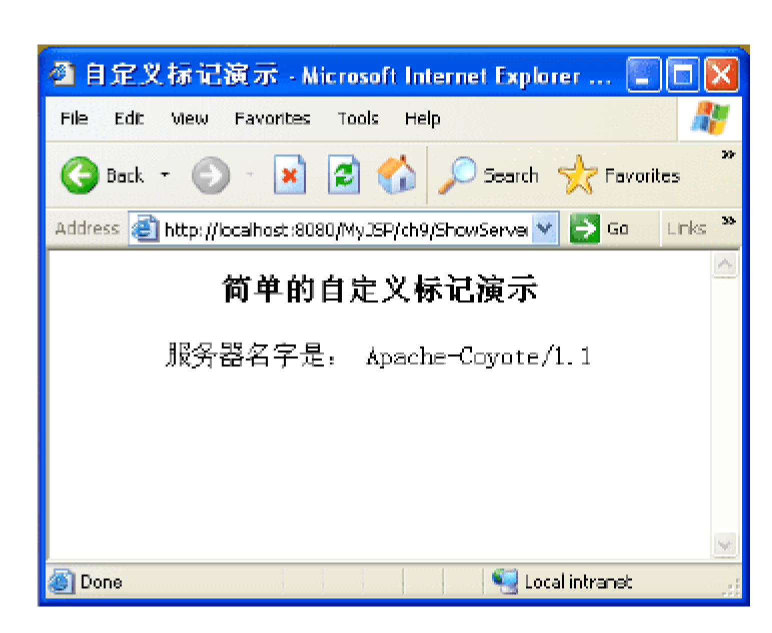


图 9-1　getWebServer 标记的输出页面

9.2.2　标记库描述符文件

标记库描述符(TLD)文件把 JSP 文件中使用的标记名和标记处理类匹配起来，该文件是 XML 格式的文件。下面的程序清单是 getWebServer 标记的 TLD 文件：

```
//test.tld
<?xml version="1.0" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2<jsp-version>
  <short-name>demo</short-name>
  <tag>
    <name> getWebServer </name>
    <tag-class>demo.getWebServerTag </tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

编写 TLD 文件需要用到 XML 的相关知识，关于 XML 的基本知识可以查看本书第 13 章或者其他相关资料。每个 TLD 文件的开始都必须声明 XML 版本和 TLD 规范的版本，供 TLD 文档处理器检验 XML 的合法性。

标记库由<taglib>元素定义，该元素首先声明标记库版本为 1.0，JSP 版本为 1.2。<taglib>元素的第 3 个子标记<short-name>是可选项，它定义了标记库的默认前缀，实际使用的前缀由 JSP 页面中的<%@taglib>指令决定，该指令定义的前缀可以不同于默认前缀。

定义了标记库信息后，TLD 文件的剩余部分是对自定义标记的描述，一个<tag>标记定义一个自定义标记，一个标记库可以包含多个自定义标记，每个自定义标记至少要含有以下内容：

标记名<name>，供页面使用；标记处理类<tag-class>，对标记进行处理；标记体类型<body-content>。在上面的例子中，标记处理类为 demo.getWebServerTag，标记名为 getWebServer，<body-content>定义了标记体的类型为空，<body-content>的可能取值如表 9-1 所示。

表 9-1 标记体类型

类　　型	描　　述
empty	空标记，如果起始标记和结束标记之间有内容，会引起代码转换错误
JSP	标记体为 JSP 文本，将和页面上的 JSP 文本一起处理
tagdependent	起始标记和结束标记间的内容将由 Java 类进行处理，不能当作 JSP 文本

9.2.3 编写标记处理类



自定义标记的标记处理类必须实现 javax.servlet.jsp.tagext 包中的接口之一，如表 9-2 所示。

表 9-2 标记处理类实现的接口

接　　口	描　　述
Tag	简单标记接口，不需要处理标记体
IterationTag	迭代标记接口，需要循环处理标记体
BodyTag	普通标记接口，仅处理标记体一次

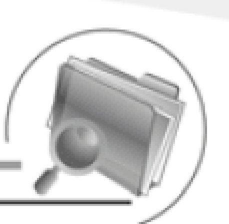
实现这些接口要求定义一些方法来管理自定义标记的生命周期。为了简化自定义标记的开发过程，javax.servlet.jsp.tagext 包包含了两个抽象类，类中提供了这些接口方法的默认行为，用户只需重载这些方法即可。表 9-3 列出了这两个类的描述。

表 9-3 实现自定义标记接口的类

类	描　　述
TagSupport 实现 Tag 接口	标记体为空或不需要处理标记体
BodyTagSupport 实现 Tag、IterationTag 和 BodyTag 接口	标记体需要被处理，如迭代标记或需要解析标记体的标记

下面是 getWebServer 标记的标记处理类的完整代码清单：

```
//getWebServerTag.java
package demo;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import javax.servlet.http.*;
import java.net.*;
```



```
public class getWebServerTag extends TagSupport {
    public int doStartTag()throws JspException{
        try{
        HttpServletRequest request =
          (HttpServletRequest)pageContext.getRequest();
        URL url = new URL("http",request.getServerName(),
            request.getServerPort(),"/");
        URLConnection conn = url.openConnection();
        String serverName = conn.getHeaderField("server");
        JspWriter out = pageContext.getOut();
        out.print(serverName);
      }catch(Exception e){
        throw new JspTagException("getWebServerTag: " + e);
      }
      return SKIP_BODY;
    }
    public int doEndTag(){
      return EVAL_PAGE;
    }
    public void release(){
    }
}
```



9.2.4　自定义标记的生命周期

自定义标记的生命周期主要有以下几种方法。

◉ doStartTag()方法

doStartTag()方法在处理起始标记时调用，返回一个 int 值表明对标记体的处理方式。

Tag.SKIP_BODY：表示忽略标记体。TLD 文件需定义标记的<body-content>为 empty。

Tag.EVAL_BODY_INCLUDE：表示标记体作为 JSP 页面的一部分进行处理。TLD 文件需定义标记的<body-content>为 JSP 或者 tagdependent。

◉ doEndTag()方法

doEndTag()方法在标记体结束时调用，返回一个 int 值表明对 JSP 剩余页面的处理方式。

Tag.EVAL_PAG：表示继续执行 JSP 文件的内容。

Tag.SKIP_PAGE：表示停止执行。

◉ doInitBody()方法

doInitBody()方法在调用 doStartTag()之后，处理标记体之前被调用。只有实现 BodyTag 接口的标记才需要。如果 doStartTag()方法返回 SKIP_BODY，则 doInitBody()方法不被调用。

◉ doAfterBody()方法

在处理完标记体之后，调用 doEndTag()之前调用该方法。

该方法的返回值代表对标记体的处理方式。

IterationTag.EVAL_BODY_AGAIN：表示需要再次处理标记体。

Tag.SKIP_BODY：表示结束对标记体的处理。

◉ release()方法

release()方法在自定义标记处理完之后被调用，释放处理占用的资源。

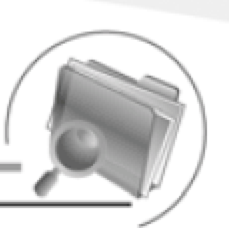
9.3 带属性的标记

标记的属性可以接收页面传递的信息，定制标记行为。属性由属性名/属性值配对组成，属性值必须用单引号或双引号括起来。

9.3.1 标记处理类

带属性的标记需要提供 set 方法对属性进行操作，程序 setAtrributeTag.java 是 setAttribute 标记的标记处理类：

```
setAtrributeTag.java
package demo;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import javax.servlet.http.*;
import java.net.*;
public class setAttributeTag extends TagSupport {
  private String param1;
  private String param2;
  private String param3="默认值";
  public void setParam1(String param1){
    this.param1 = param1;
  }
  public void setParam2(String param2){
    this.param2 = param2;
  }
  public void setParam3(String param3){
    this.param3 = param3;
  }
  public int doStartTag()throws JspException{
    try{
      JspWriter out = pageContext.getOut();
```



```
        out.print(param1+"<br>");
        out.print(param2+"<br>");
        out.print(param3);
      }catch(Exception e){
        throw new JspTagException(e.getMessage());
      }
      return SKIP_BODY;
    }
    public int doEndTag(){
      return EVAL_PAGE;
    }

}
```

每个属性的 set 方法都会在 doStartTag()方法之前被调用。

9.3.2　标记库描述符文件



TLD 文件必须对标记的所有属性进行定义。每个属性名必须与其细节信息一起列出，它必须包含一个<attribute>属性，以及该属性所带的子元素，如表 9-4 所示。

表 9-4　标记属性的 TLD 描述

元　素	描　述
attribute	在 TLD 的<tag>组件中引入标记的属性定义
name	不可省略的元素，定义属性名
required	不可省略的元素，true 表示属性必须存在，false 表示属性可有可无
rtexprvalue	定义属性值是否允许动态表达式
type	属性类型，默认为 String，当 rtexprvalue 为 true 且该属性类型不为 String 时必须定义

程序 SetAttribute.jsp 中的 setAtrrib 标记在 TLD 中的定义如下：

```
<tag>
  <name>setAtrrib</name>
  <tag-class>demo.setAttributeTag</tag-class>
  <body-content>empty</body-content>
  <attribute>
     <name>param1</name>
     <required>true</required>
   </attribute>
   <attribute>
      <name>param2</name>
      <required>false</required>
```

```
        </attribute>
        <attribute>
            <name>param3</name>
            <required>false</required>
        </attribute>
</tag>
```

9.3.3 使用标记

下面看一个简单的实例，该实例改进了 9.3.2 节中的例子，增加了标签属性。

```
//SetAttribute.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<%@ taglib prefix="demo" uri="/WEB-INF/test.tld"%>
<html>
    <head>
        <title>带属性的标记演示</title>
    </head>
    <body bgColor="skyblue">
    <h3>带属性的标记演示</h3>
    <h4>
        <demo:setAtrrib param1="参数一" param2="参数二"/>
    </h4>
    </body>
</html>
```

运行该实例，结果如图 9-2 所示。

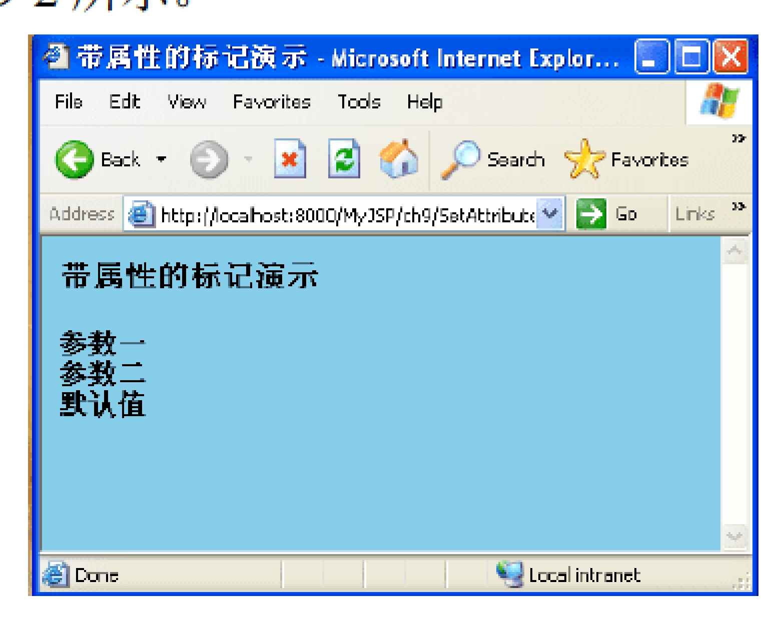


图 9-2　带属性的标记演示

9.4 嵌入标记主体的标记

到目前为止，本书介绍的自定义标记都忽略了标记主体，使用的都是单独形式的标记：

```
<prefix:tagname />
```

本节将介绍如何创建使用主体内容的标记，如下面的方式：

```
<prefix:tagname>body</prefix:tagname>
```

9.4.1　标记处理类

在 9.3.1 节的例子中，标记处理类定义了一个 doStartTag 方法，其返回值为 SKIP_BODY。为了让系统使用起始标记和终止标记中出现的主体，doStartTag 方法必须返回 EVAL_BODY_INCLUDE 值。这个主体内容可以包含 JSP 脚本元素、指令和行为。如果使用了标记主体，则可以使用 doEndTag 方法在主体后做些处理动作。几乎所有情况下，用户要在标签结束后继续输出和处理页面，因此 doEndTag 方法应当返回 EVAL_PAGE；如果要停止处理剩下的页面，则可以返回 SKIP_PAGE。 下面来看一个例子：

```
HeadingTagExample.java
package demo;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
public class HeadingTag extends TagSupport {
  private String bgColor; // The one required attribute
  private String color = null;
  private String align="CENTER";
  private String fontSize="36";
  private String fontList="Arial, Helvetica, sans-serif";
  private String border="0";
  private String width=null;
  public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
  }
  public void setColor(String color) {
    this.color = color;
  }
  public void setAlign(String align) {
    this.align = align;
  }
  public void setFontSize(String fontSize) {
    this.fontSize = fontSize;
  }
  public void setFontList(String fontList) {
```

```
    this.fontList = fontList;
  }
  public void setBorder(String border) {
    this.border = border;
  }
  public void setWidth(String width) {
    this.width = width;
  }
  public int doStartTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("<TABLE BORDER=" + border +
        " BGCOLOR=\"" + bgColor + "\"" +
        " ALIGN=\"" + align + "\"");
      if (width != null) {
        out.print(" WIDTH=\"" + width + "\"");
      }
      out.print("><TR><TH>");
      out.print("<SPAN STYLE=\"" +
        "font-size: " + fontSize + "px; " +
        "font-family: " + fontList + "; ");
        if (color != null) {
        out.println("color: " + color + ";");
        }
      out.print("\"> "); // End of <SPAN ...>
    } catch(IOException e) {
      System.out.println(e.getMessage());
    }
      return(EVAL_BODY_INCLUDE); // Include tag body
  }
    public int doEndTag() {
    try {
      JspWriter out = pageContext.getOut();
      out.print("</SPAN></TABLE>");
    } catch(IOException e) {
      System.out.println(e.getMessage());
    }
    return(EVAL_PAGE);
  }
}
```



9.4.2　标记库描述符文件

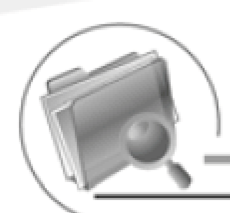
对于使用主体的标记，在标记元素里只有一个新特征：bodycontent 元素应当像下面这样包含 JSP 值。

```
<body-content>JSP</body-content>
```

name、tag-class 和 attribute 元素跟前面的使用一样，如下所示：

```
<tag>
  <name>heading</name>
  <tag-class>demo.HeadingTag</tag-class>
  <body-content>JSP</body-content>
  <attribute>
    <name>bgColor</name>
    <required>true</required> <!-- bgColor is required -->
  </attribute>
  <attribute>
    <name>color</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>align</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>fontSize</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>fontList</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>border</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>width</name>
    <required>false</required>
  </attribute>
 </tag>
```





9.4.3 使用标记

程序 HeadingTagExample.jsp 使用了刚刚定义的 heading 标签。因为 bgColor 属性被定义为必需的，所有使用这个标签时都要包含它。程序清单如下：

```
HeadingTagExample.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<%@ taglib prefix="demo" uri="/WEB-INF/test.tld"%>
<HTML>
 <HEAD>
   <TITLE>带主体的标记演示</TITLE>
 </HEAD>
 <BODY>
   <demo:heading bgColor="#C0C0C0">
     标题样式 1
   </demo:heading>
   <p>
   <demo:heading bgColor="BLACK" color="WHITE">
标题样式 2
   </demo:heading>
   <p>
   <demo:heading bgColor="yellow" fontSize="50" border="5">
     标题样式 3
   </demo:heading>
   <P>
   <demo:heading bgColor="green" width="50%">
     标题样式 4
   </demo:heading>
   <P>
   <demo:heading bgColor="skyblue" fontSize="40"
     fontList="Brush Script MT, Times, serif">
     标题样式 5
   </demo:heading>
   <P>
 </BODY>
</HTML>
```

运行该实例，结果如图 9-3 所示。





图 9-3　嵌入标记主体的标记演示



9.5　定义脚本变量的标记

JSP 页面的开发者对脚本变量都很熟悉：它们通常是定义在一个 scriptlet 或<jsp:useBean>行为中的 Java 变量。例如，下面代码开始处的 scriptlet：

```
<%
   String[] flavors = {"Chocolate", "Strawberry", "Vanilla" };
   for( int i=0; i<flavors.length; i++ ){
%>
<h3> Flavor <%= i %> is <%=   flavors[i] %> </h3>
<%
   }
%>
```

这里定义的整型变量 i 和字符串数组变量 flavors 可用于页面上其他 scriptlet 和表达式。

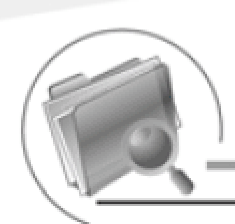
再看看<jsp:useBean>行为中的 Java 变量：

```
<jsp:useBean id="id1"   class="Meteor">
   <jsp:setProperty name="id1" property="bane" value="Atmosphere" />
</jsp:useBean>
```

显示变量值可以用<%= id1.getBane() %>。

<jsp:useBean>行为定义了类 Meteor 名为 id1 的变量。它被紧接着的<jsp:setProperty>行为使用，并被最后一行的显示变量值的表达式使用。

定制标记也可以在其标记处理器中定义脚本变量，并可用于 scriptlet、表达式和同一页面上的其他标记。



可以通过以下两种方式在标记库描述符(TLD)文件中定义脚本标记：

⊙ 在 TLD 文件中列明脚本变量

一个<variable>元素定义一个脚本变量，它必须包含表 9-5 中列出的子元素。

表 9-5　TLD 文件中定义脚本变量<variable>元素的子元素

元　素	描　述
name-given	定义变量名(不能与 name-from-attribute 同时使用)
name-from-attribute	指定属性值作为变量名(不能与 name-given 同时使用)
variable-class	指定脚本变量的类
declare	指定脚本变量是否为新建的对象(默认为 true)
scope	变量的作用范围：即页面的哪部分可以操作这个变量，取值为 NESTED、AT_BEGIN 或 AT_END。 NESTED：变量在 starttag 到 endtag 之间是有效的。 AT_BEGIN：变量在标签的开始到 JSP 页面结束是有效的。 AT_END：变量在标签的结束到 JSP 页面结束是有效的



定义字符串变量的 TLD 描述如下：

```
<variable>
  < name-given >title< /name-given>
  <variable-class>java.lang.String</ variable-class>
  <declare>true<.declare>
  <scope>AT_BEGIN</scope>
</variable>
```

⊙ 定义标签扩展信息类(TEI)并且在 TLD 文件中包括这个类元素<tei-class>

在<tag>元素中包含子元素<tei-class>，例如：

```
<tei-class>demo.DefineVar</tei-class>
```

使用第一种方式虽然很方便，但是不够灵活，下面主要介绍通过标签扩展信息类来定义脚本变量的标签。

9.5.1　类 TagExtraInfo

通过扩展类 javax.servlet.jsp.TagExtraInfo 可以定义一个额外标签信息类，使用<tei-class>元素在 TLD 中指定这个扩展类。JSP 容器转换时在提供的 TagExtraInfo 类上调用 getVariableInfo()方法，它将传递一个 javax.servlet.jsp.tagext.TagData 对象，该对象包含在转换时识别的[属性，值]对。JSP 容器使用 TagData 实例向 TagExtraInfo 类提供这些属性及其值。下面的例子中自定义标记<sample:VarTag>指定了 3 个脚本变量 Year、Principal 和 Balance，变量名通过标记的属



性值来提供：

```
<sample:VarTag yearName="Year" principalName="Principal" balanceName="Balance" >
```

getVariableInfo()方法返回一个 javax.servlet.jsp.tagext.VariableInfo 对象的数组。每一个 VariableInfo 对象包含 4 个参数。

下面看一个自定义标记的 TagExtraInfo 类：

```
public class VarTagTEI extends TagExtraInfo {
 public VariableInfo[] getVariableInfo(TagData data){
   VariableInfo info1 = new VariableInfo(data.getAttributeString("yearName "),
      "String",true,VariableInfo.NESTED);
   VariableInfo info2 = new VariableInfo(data.getAttributeString("principalName "),
      "String",true,VariableInfo.NESTED);
   VariableInfo info3 = new VariableInfo(data.getAttributeString("balanceName "),
      "String",true,VariableInfo.NESTED);
   VariableInfo info[] = {info1,info2,info3};
   return info;
 }
}
```



VarTagTEI 类定义了 3 个脚本变量。VariableInfo 构造函数的第 1 个参数通过 TagData 对象检索属性名称对应的值，返回变量名；第 2 个参数指出变量的类型为 String；第 3 个参数指明该变量对于 JSP 页面来说是否为新的；第 4 个参数定义了变量的作用域，脚本变量有 3 种可能的作用域：VariableInfo.NESTED——可以在定义脚本变量的标记开始和结束之间使用该变量，VariableInfo.AT_BEGIN——可以从定义脚本变量的标记开始到页面结束之间使用该变量，VariableInfo.AT_END——从定义脚本变量的标记结束之后到页面结束之间使用该变量。

9.5.2　定义脚本变量

标记处理程序定义了脚本变量，它在 TagExtraInfo 类中指定的确切名称下面使用 setAttribute(name,value)方法将变量存储在 pageContext 中。JSP 容器在 TagExtraInfo 类中创建一个脚本变量，并将它赋给在 pageContext 中的同一名称下面存储的对象。

9.5.1 节中，自定义标记<sample:VarTag>指定了 3 个脚本变量 Year、Principal 和 Balance。因此，标记处理程序必须创建这些变量并使用 pageContext 对象存储它们。下面的代码段显示了如何存储这 3 个变量：参数 value 用于设置变量的值。

```
pageContext.setAttribute("Year",value);
pageContext.setAttribute("Principal",value);
pageContext.setAttribute("Balance",value);
```

9.5.3 典型实例

首先看一个 JSP 代码清单：

```
DefineVar.jsp
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<%@ taglib prefix="demo" uri="/WEB-INF/test.tld"%>
<demo:defineVariable name="name" type="String" />
<demo:defineVariable name="age" type="String" />
<demo:defineVariable name="sex" type="String" />
<HTML>
 <HEAD>
   <TITLE>定义变量的标记演示</TITLE>
 </HEAD>
 <BODY bgcolor="skyblue">
   <CENTER>
   <h3>定义变量的标记演示</h3>
   <table border="1">
     <tr>
       <th>变量名</th>
       <th>变量值</th>
     </tr>
     <tr align="center">
       <td width="100" >姓名</td>
       <td width="100"><%= name %></td>
     </tr>
     <tr align="center">
       <td>年龄</td>
       <td><%= age %></td>
     </tr>
     <tr align="center">
       <td>性别</td>
       <td><%= sex %></td>
     </tr>
   </table>
   </CENTER>
 </BODY>
</HTML>
```

该 JSP 页面 3 次调用自定义标记 defineVariable，这个标记使用了 2 个属性，即脚本变量名



称和类型。在处理标记之后，JSP 页面可以访问 3 个隐含变量：String 类型的变量 name、age 和 sex。使用 JSP 表达式在 HTML 表格中显示这些变量的值。

◉ 创建标记库描述文件

TLD 文件的内容如下：

```
<?xml version="1.0" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
 <tag>
  <name>defineVariable</name>
  <tag-class>demo.defineVarTag</tag-class>
  <tei-class>demo.defineVarTEI</tei-class>
  <body-content>empty</body-content>
  <attribute>
   <name>name</name>
   <required>true</required>
  </attribute>
  <attribute>
   <name>type</name>
   <required>true</required>
  </attribute>
 </tag>
</taglib>
```



该 TLD 文件说明了名为 defineVariable 的自定义标记包含 Java 类 demo.defineVarTag 作为它的标记处理程序，包含 demo.defineVarTEI 类作为它的 TagExtraInfo 类。标记需要包括一个空白主体并使用 2 个属性：name 和 type，而且这 2 个属性都是必需的。

◉ 创建自己的 TagExtraInfo 类

编写自己的 TagExtraInfo 类，创建名为 defineVarTEI.java 的源程序，代码清单如下：

```
defineVarTEI.java

package demo;
import javax.servlet.jsp.tagext.*;

public class defineVarTEI extends TagExtraInfo {
 public VariableInfo[] getVariableInfo(TagData data){
  VariableInfo info1 = new VariableInfo(data.getAttributeString("name"),
     data.getAttributeString("type"),true,VariableInfo.AT_END);
```




```
    VariableInfo info[] = {info1};
    return info;
  }
}
```

类派生于 TagExtraInfo 类，定义了 getVariableInfo 方法，它描述了由这个自定义标记创建的每一个脚本变量。这里指定了 VariableInfo.AT_END 作用域，表明 JSP 页面在自定义标记结束后使用变量。

◉ 创建标记处理程序

标记处理程序 defineVarTag.java 的程序清单如下：

```
package demo;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class defineVarTag extends TagSupport {
  private String name = null;
  private String type = null;
  public void setName(String name){
    this.name = name;
  }

  public void setType(String type){
    this.type = type;
  }

  public int doEndTag() throws JspException{
    if(name.equals("name"))
      pageContext.setAttribute("name","李小鹿");
    else if(name.equals("age"))
      pageContext.setAttribute("age","25");
    else if(name.equals("sex"))
      pageContext.setAttribute("sex","女");
    return EVAL_PAGE;
  }
}
```

因为自定义标记包括一个空白主体，所以标记处理程序派生于 TagSupport。由于脚本变量具有 VariableInfo.AT_END 作用域，所以需要能够在标记处理程序的 doEndTag()方法中使用。

运行 DefineVar.jsp，显示结果如图 9-4 所示。





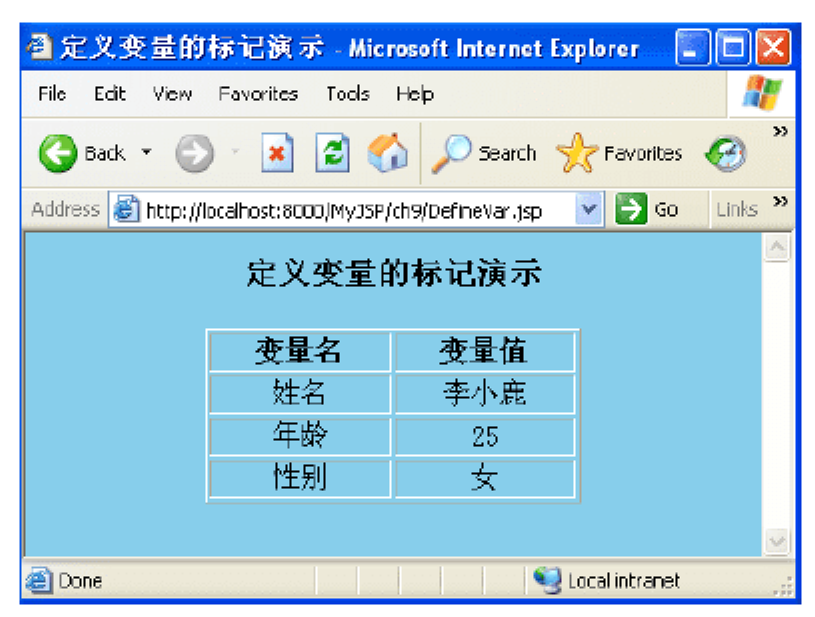


图 9-4　定义变量的标记演示

9.6　上机练习



本章上机实验主要练习如何创建自定义的 JSP 标记以及如何在 JSP 中使用这些自定义的标记进行编程。其中重点掌握如何编写自定义标记处理类、如何编写标记库描述符文件、如何使用自定义标记等操作。

下面以在简单的自定义标记中获取服务器名为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮，Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中将出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Class】命令，弹出 New Java Class 对话框新建自定义标记处理类。

(7) 在 New Java Class 对话框的 Package 文本框中输入 Java 类所使用的包名，例如 ch9。

(8) 在 Name 文本框中输入 Java 类的名字，如 getWebServerTag.java。在 Superclass 文本框中输入 javax.servlet.jsp.tagext.TagSupport。

(9) 单击 Finish 按钮完成 Java 类的创建。

(10) Eclipse 会创建一个包括了基本类框架的新类。在类的编辑窗口中输入 getWebServerTag.java 程序代码即可。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 选择【Project】|【Build All】命令，编译 getWebServerTag.java 文件。

(14) 检查工作空间 testTomcat 项目 WEB-INF 下的 classes 文件夹，可以找到相应的 ch9/getWebServerTag.class 文件。

(15) 使用【记事本】或者其他 XML 文件编辑器编写 TLD 文件，以 test.tld 为名保存于 WEB-INF 目录下。

(16) 选择【File】|【New】|【JSP】命令，弹出 New JavaServer Page 对话框创建一个 JSP 文件。

(17) 在 New JavaServer Page 对话框的 File name 文本框中输入文件名，如 ShowServer.jsp。

(18) 单击 Finish 按钮，新建 JSP 页面文件。

(19) Eclipse 会创建一个新的 JSP 文件，包括基本的页面框架。在编辑窗口中输入 ShowServer.jsp 页面代码即可。

(20) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，将显示错误信息。

(21) 根据错误提示，修改所有可能的语法错误。

(22) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(23) 打开 Internet Explorer 浏览器，输入对应的 URL，如：http://localhost:8080/testTomcat/ch9/ShowServer.jsp。观察页面是否正常运行。

9.7 习题

9.7.1 填空题

1. 支持自定义标记的 Java 包是________________。
2. 自定义标记通常继承的抽象类是________________和________________。
3. 脚本变量的作用域有________________、________________和________________。

9.7.2 选择题

1. 下面哪一个不是标记体的类型(　　　)。

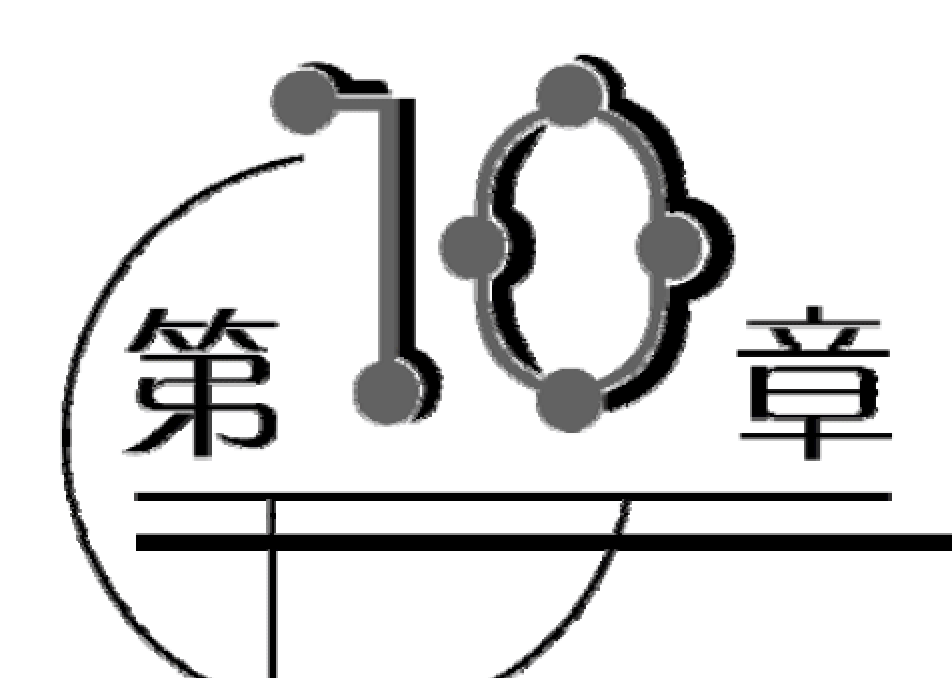
A. empty　　B. JSP　　C. taglib　　D. tagdependent

2. 标记处理类需要实现的接口包括(　　　)(多选)。

A. Tag　　B. IterationTag　　C. BodyTag　　D. jspTag

9.7.3 问答题

1. TLD 是什么？起什么作用？
2. 可以重载 BodyTagSupport 类有 5 种方法，在处理标记时按什么顺序调用这些方法？



第10章 JSP 安全性

学习目标

本章将介绍 JSP 的安全结构，这个主题复杂多样，本书只作简要介绍。应用程序安全性尝试保护信息免受非法访问或修改，管理公司或者个人信息的部署级 JSP 应用程序必须保护这种信息。本章提供了安全性基础知识的概述，然后介绍用于 JSP 结构和基于 Java 的 Web 应用程序的特定安全过程。

本章重点

- 应用程序的安全性
- Web 认证
- Servlet 容器认证

10.1 基本应用程序安全性

在建立应用程序时必须考虑以下几项安全服务：认证、授权、机密性、完整性、不可否认以及审计。其中用户认证是这些安全服务中最基本的。

1. 认证

认证就是识别一个客户是否为系统的合法用户。识别一个客户包括两部分：初始地确认客户身份；每次客户访问应用程序时进行客户认证。

- 初始识别

在最简单的级别上，初始识别要求用户简单地注册应用程序而不用其他鉴定。比较常见的是：公司人力资源部门或管理人员识别一个用户。被鉴定通过的用户被系统注册并准予访问系统。

◉ 客户认证

应用程序的注册用户每次使用应用程序时都必须标识自己的身份。最常见的认证形式是给每个用户一个惟一的名字(典型的是一个账号或登录名)和与此账号对应的密码。用户必须简单地提供账号和密码才能访问应用程序。

标识一个客户的信息通常叫作用户凭证，最常见的用户凭证形式有：账户名和密码、读卡、智能卡以及数字证书。

认证就像在公园的大门入口，只要有一张门票就被允许进入公园，即已经被认证。但是认证不表示可以使用公园里的所有设备，允许接触公园不同的地方叫做授权。

2. 授权

授权包括根据被认证用户的身份来控制该用户对应用程序的访问。不同类别或类型的用户被授予不同的权限，允许或拒绝他们访问系统的不同部分。这也类似于在公园，可能仅被授权使用特定的乘骑：乘骑有一定的高度、重量或年龄限制来允许一些用户使用。

3. 机密性

安全的另一个方面不是控制对功能的访问，而是保证数据只对被授权的用户可见，换句话说就是保证数据的机密。保证数据的机密不仅是授权对数据的访问，而且也要保证未授权的访问不会发生。在实际操作中，机密性是通过加密数据实现的，这样，只有被授权的用户才能解密访问数据。

4. 完整性

保证数据的完整性也就是防止数据被蓄意或意外地以一种未授权的方式修改。使用授权恰当地解决了在访问服务数据时大部分数据的完整性问题。数据在网络传输过程中不能改变和破坏，用户必须确信他们收到的数据是被传输过来的数据。很多技术(例如加密、验证和数字签名)等都能保证数据在穿越网络时是完整的。

完整性也意味着对系统所做的任何改变都不会丢失，可是当服务器崩溃时，丢失情况就可能发生。好的审计实务(见后面的【审计】部分)就可以帮助用户防止持久性数据改变的丢失。

5. 不可否认

不可否认的意思是指能够证明一个用户做过某件事情，即便该用户随后否认。举一个简单的例子：如果一个用户将自己账户中的钱转移到另一个银行账户，然后宣称自己没做过，是银行系统的错误。如果拥有好的账目处理过程，银行就能证明事情并不是这样的。

6. 审计

审计对数据库用户来说并不陌生，安全中的审计也是一样的意思，它提供活动的记录。好的审计是支持不可否认和完整性的助手，但是审计必须是安全的。



10.2　Web 认证

在 Web 应用程序中有许多用于认证的技术，本节主要介绍 3 种技术：LDAP 认证、基于证书的认证、基于 Web 服务器的认证。LDAP 认证和基于证书的认证对许多现代应用程序具有普遍的重要性，而基于 Web 服务器的认证只适用于 Web 应用程序。

10.2.1　LDAP 认证

协作认证方案允许应用程序共享安全信息，所以它们具有明显的实用性。例如：网络目录服务提供了包含有效用户和所需的认证信息的中央数据库。这意味着用户只需登录一次就可以访问多个应用程序。由于局域网(Local Area Network，LAN)的出现以及建立一个网络身份而不是在一台单独的计算机建立单独身份的要求，使得这种服务的提供成为可能。



LDAP 服务器实现了轻量级目录访问协议(Lightweight Directory Access Protocol, LDAP)以提供一项通用的目录服务。LDAP 服务器将信息存储为树结构中的项目。对象类定义了每一个项目的类型，包括必需属性和可选属性。可分辨名称(Distinguished Name，DN)惟一标识了各个项目。DN 的格式包含项目的标记加上到树的根目录的路径。如图 10-1 所示，显示了一个 LDAP 目录结构。

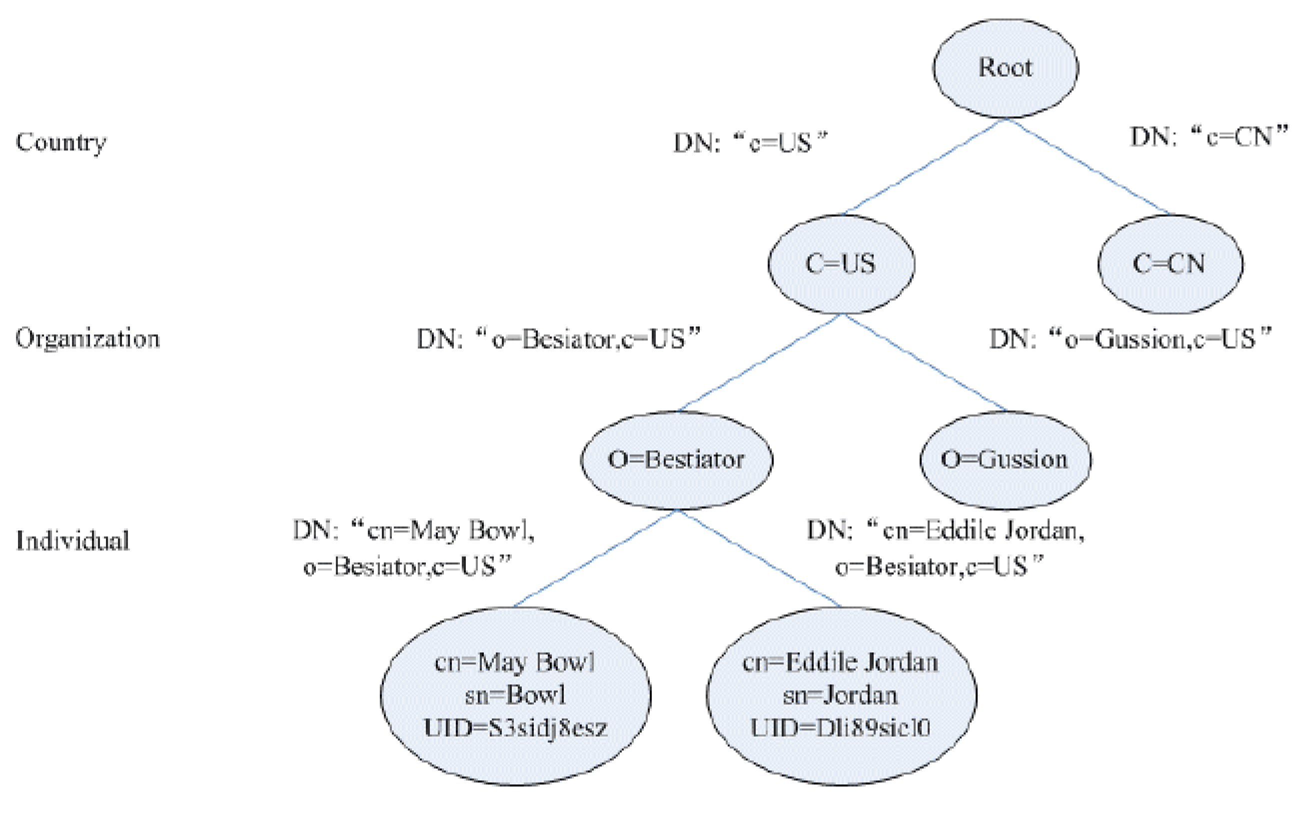


图 10-1　LDAP 目录结构

图 10-1 显示了 3 个 LDAP 对象类：Country、Organization 和 Individual。Country 项目填充了 LDAP 分层结构中的第 1 级，这些项目有一个单独的必需属性 c (代表 country)。Organization 项目填充了分层结构的第 2 级，这些项目有一个单独的必需属性 o (代表 organization)。Individual 项目填充了分层结构的树叶注释。这些项目具有 2 个必需属性：cn(代表 common name)和 sn(代表 surname)。它们还有一个可选属性 UID(代表 user id)。为了检索像 UID 这样的应用程序属性，应用程序将向 LDAP 服务器提供认证信息。

10.2.2 基于证书的认证

基于证书的认证依赖于 X.509 安全证书。像 LDAP 安全认证方案一样，基于证书的安全方案也有助于集中安全信息。不过，它的主要优点是加密：这样的证书使用公钥/私钥加密技术建立身份。下面进一步讨论它是如何工作的。

传统的消息加密依赖于消息的发送者和接收者都知道的一个公共密钥。在这种方案(称为密钥或对称加密)中，发送者使用该密钥加密消息并发送给接收者，接收者使用相同的密钥解密消息。虽然这种方案发送者和接收者在相同的物理位置时工作得很好，但在像万维网这样的广泛分布空间中，它工作得并不是很好。主要的问题是共享同一个密钥并且要保密，共享一个密钥的人越多，它的保密性就越差，保密也就越困难。因此这个密钥管理问题引出了公钥方案。

公钥方案通过将加密分解为 2 个密钥(一个公钥和一个私钥)简化了保密工作。用户只公布自己的公钥，使私钥保密。私钥和公钥是成对工作的大数字，用来加密和解密数字签名。实际上，公钥/私钥加密技术背后的数学知识确保了不可能从一个公钥猜测到它的私钥。用户用私钥加密消息，消息的接收者使用公钥来解密消息。类似地，接收者用公钥加密要返回的消息，用户使用私钥解密。

X.509 证书存储了一个实体的身份，并通过使用该实体的私钥(也就是通过创建一个摘要)加密某段数据以数字方式签署它。已签署消息的接收者必须使用这个实体的公钥解密签名。只有当签名有效时(也就是说，只有当客户使用匹配的私钥加密这个签名时)，才能解密成功。实体本身只知道私钥，如果解密成功则表示签名是有效的，实体得到认证。如图 10-2 所示，显示了数字签名认证实体的过程。

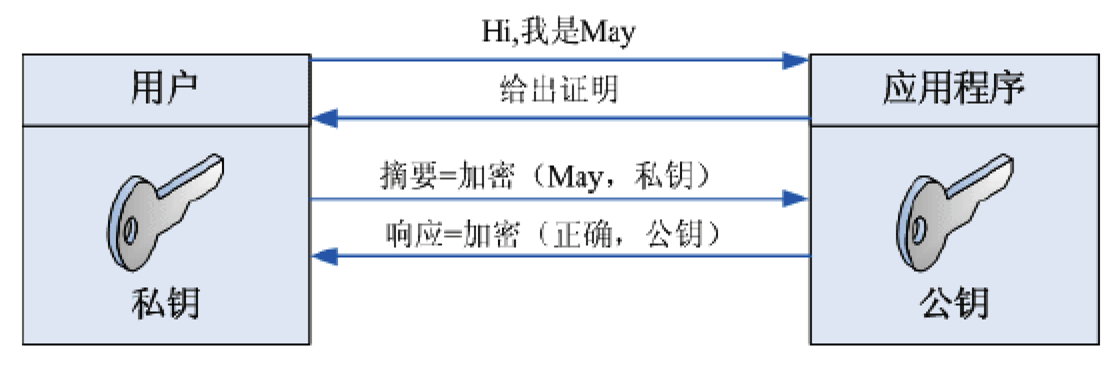


图 10-2 基于数字证书的认证

目前，Web 浏览器是安全证书的最著名用户，它使用证书来启用安全套接字层(Secure Socket Layer，SSL)协议。Web 站点也使用证书来将自己标识为浏览器用户。Java 程序员可以使用 java.security.cert 程序包并从开发商处购买在这个程序包中定义的抽象类的实现，从而在应用程序中嵌入基于证书的认证。

10.2.3 基于 Web 服务器的认证

每一台 Web 服务器都提供了某种机制来控制资源的访问。本书的例子是 Apache Web 服务器(Apache)。Apache 支持基本认证和摘要认证两种形式。与基于证书的认证类似，基本认证和



摘要认证使用质询-响应协商的形式。如图 10-3 所示，说明了这种认证机制的工作原理。



图 10-3　质询-响应安全机制

在质询-响应协商中，客户首先请求访问特定的资源。如果对资源的访问受到控制，那么安全机制将截取这个请求并返回一个授权错误。浏览器将显示一个对话框，用户必须输入凭据信息。如果客户提供了足够的凭据，那么安全机制将允许这个请求；否则，安全机制将返回另一个授权错误。如果授权成功，浏览器通常会缓存认证信息，这样客户在会话持续期间就不需要再次提供这些认证信息。

摘要认证将凭据的单向散列添加到消息摘要中(也就是说，以一种不可读的形式进行用户名和密码的不可逆编码)。这允许用户输入用户名和密码，以认证自己，但是不以明文形式在网络上发送这些凭据。摘要认证最严重的弱点是没有提供向服务器传递摘要的安全方式，第二个问题是这种模式没有提供任何方法在用户与特定用户的正确摘要的服务器之间建立初始排列。



10.3　Servlet 容器认证

Servlet 容器认证只与 Servlet 和 JSP 应用程序有关。Servlet 2.2 规范鼓励和要求 4 种认证方法：基本认证、摘要认证、基于表单的认证和使用 HTTPS 客户认证。下面就详细讨论每一种方法。

10.3.1　基本认证

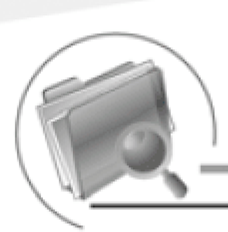
Servlet 容器的基本认证与 Web 服务器(如 Apache)中的基本认证完全一样。Servlet 2.2 规范需要所有与 Java 2 Enterprise Edition(J2EE)兼容的 Servlet 容器都支持认证。除了其他优点之外，还允许像 Tomcat 这样的 Servlet 容器以一种轻量级的独立模式工作，供 Web 站点开发人员使用。

下面介绍配置 Tomcat 的基本认证，Tomcat Servlet 容器中的基本认证需要 3 个步骤：

(1) 向%TOMCAT_HOME%\conf 中的 tomcat-users.xml 文件中的<tomcat-users>模块添加用户、密码和角色。

(2) 将<security-constraint>模块添加到应用程序的 web.xml 文件中，以便在该应用程序环境中配置它。

(3) 将<login-config>模块添加到应用程序的 web.xml 文件中，以便在该应用程序环境中配置它。



1. 配置用户、密码和角色

Tomcat 的安装提供了一些用户和角色以支持它的例子。在默认的 tomcat-users.xml 文件中可以看到<tomcat-users>模块：

```
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="" roles="admin,manager"/>
</tomcat-users>
```



这个<tomcat-users>模块定义了 4 个用户：tomcat、both、role1 和 admin。前 3 个用户共享密码 tomcat，用户 admin 的密码为空。第 1 个用户属于角色 tomcat，第 2 个用户属于角色 tomcat 和 role1，第 3 个用户属于角色 role1，第 4 个用户属于角色 admin 和 manager。

2. 配置<security-constraint>模块

在添加用户之后，必须在应用程序的 web.xml 文件中配置<security-constraint>模块。Tomcat 提供了一个配置安全性的示例应用程序，可以在%TOMCAT_HOME%\ webapps\jsp-examples 目录中找到。在这个应用程序的 web.xml 文件(该文件在 WEB-INFO 目录中)中可以看到<security-constraint>模块：

```
<security-constraint>
    <display-name>Example Security Constraint</display-name>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
          <!-- 定义受保护的页面 -->
        <url-pattern>/security/protected/*</url-pattern>
          <!-- 举受保护的 HTTP 方法 -->
        <http-method>DELETE</http-method>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
        <http-method>PUT</http-method>
    </web-resource-collection>
    <auth-constraint>
        <!—可以访问资源的角色 -->
        <role-name>tomcat</role-name>
```



```
      <role-name>role1</role-name>
    </auth-constraint>
</security-constraint>
```

这个<security-constraint>模块使用了子元素<web-resource-collection>和<auth-constraint>来配置示例应用程序环境。<web-resource-collection>子元素确定应该保护的资源，所有<security-constraint>元素都必须包含至少一个<web-resource-collection>子元素，此元素由一个给出任意标识名称的<web-resource-name>元素、一个确定应该保护的 URL 的<url-pattern>元素、一个指出此保护所适用的 HTTP 命令(GET、POST 等，默认为所有方法)的<http-method>元素和一个提供资料的可选<description>元素组成；<auth-constraint>子元素指出哪些用户应该具有受保护资源的访问权，此元素应该包含一个或多个标识具有访问权限的用户类别<role-name>元素，以及一个描述角色的<description>元素(可选)。

<security-constraint>模块还支持第 3 个子元素：<user-data-constraint>。这个可选的子元素指出在访问相关资源时使用任何传输层保护，它必须包含一个<transport-guarantee>子元素(合法值为 NONE、INTEGRAL 或 CONFIDENTIAL)，并且包含一个可选的<description>元素。<transport-guarantee>为 NONE 值表示对所用的通信协议不加限制；INTEGRAL 表示数据必须以一种防止截取它的人阅读它的方式传送，虽然原理上，在 INTEGRAL 和 CONFIDENTIAL 之间可能会有差别，但在当前实践中，他们都只是简单地要求用 SSL。例如，下面的代码指示服务器只允许对相关资源做 HTTPS 连接：




```
<security-constraint>
  <!-- ... -->
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

3. 配置<login-config>模块

最后，必须为应用程序配置<login-config>模块。在此查看 Tomcat 示例应用程序的 web.xml 文件，可以看到如下所示的<login-config>模块：

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Example Form-Based Authentication Area</realm-name>
  <form-login-config>
    <form-login-page>/security/protected/login.jsp</form-login-page>
    <form-error-page>/security/protected/error.jsp</form-error-page>
  </form-login-config>
  </login-config>
```

使用<login-config>元素规定服务器应该怎样验证试图访问受保护页面的用户。它包含 3 个可能的子元素，分别是：<auth-method>、<realm-name>和<form-login-config>。<login-config>元素应该出现在 web.xml 部署描述符文件的结尾部分，紧跟在<security-constraint>元素之后。

<auth-method>：这个子元素列出服务器要使用的特定验证机制，有效值为 BASIC、DIGEST、FORM 和 CLIENT-CERT，服务器只需要支持 BASIC 和 FORM。 BASIC 指出应该使用标准的 HTTP 验证，在此验证中服务器检查 Authorization 头。如果缺少这个头则返回 401 状态代码和一个 WWW-Authenticate 头。这将导致客户机弹出一个用于填写 Authorization 头的对话框。此机制很少或不提供对攻击者的防范，这些攻击者在 Internet 连接上进行窥探(如通过在客户机的子网上执行一个信息包探测装置)，因为用户名和口令是用简单的可逆 base64 编码发送的，所以他们很容易得手。所有兼容的服务器都需要支持 BASIC 验证。DIGEST 指出客户机应该利用加密 Digest Authentication 形式传输用户名和口令，从而提供了比 BASIC 验证更高的防范网络截取的安全性，但这种加密比 SSL(HTTPS)所用的方法更容易破解。FORM 指出服务器应该检查保留的会话 cookie 并且把不具有它的用户重定向到一个指定的登录页。此登录页应该包含一个收集用户名和口令的常规 HTML 表单。在登录之后，利用保留会话级的 cookie 跟踪用户。虽然很复杂，但 FORM 验证防范网络窥探并不比 BASIC 验证更安全，如果有必要可以在顶层安排诸如 SSL 或网络层安全(如 IPSEC 或 VPN)等额外的保护。所有兼容的服务器都需要支持 FORM 验证。CLIENT-CERT 规定服务器必须使用 HTTPS(SSL 之上的 HTTP)并利用用户的公开密钥证书(Public Key Certificat)对用户进行验证。这为防范网络截取提供了很强的安全性，但只有兼容 J2EE 的服务器需要支持它。

<realm-name> ：此元素只在<auth-method>为 BASIC 时使用。它指出浏览器在相应对话框标题使用的、并作为 Authorization 头组成部分的安全域的名称。

<form-login-config>： 此元素只在<auth-method>为 FORM 时使用。它指定两个页面，分别是包含收集用户名及口令的 HTML 表单页面(利用<form-login-page>子元素)，和用来指示验证失败的页面(利用<form-error-page>子元素)。

10.3.2 摘要认证

摘要认证的工作方式和基本认证相似，只是用户名和密码是以加密的形式返回的。对于抵抗非法的网络通信监测，加密后的用户名和密码更安全。摘要认证没有被 Servlet 2.3 规范要求，因为它目前没有被 Web 客户端广泛支持。

10.3.3 基于表单的认证

基于表单的认证允许提供一个自定义的登录页。在%TOMCAT_HOME%\webapps\jsp-examples 示例的安全应用程序中，在 web.xml 文件中可以找到这么一段代码：

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Example Form-Based Authentication Area</realm-name>
  <form-login-config>
    <form-login-page>/security/protected/login.jsp</form-login-page>
    <form-error-page>/security/protected/error.jsp</form-error-page>
  </form-login-config>
  </login-config>
```

基于表单的认证通过如下语句向</form-login-config>模块添加登录页和错误页的说明：

```
<form-login-page>/security/protected/login.jsp</form-login-page>
<form-error-page>/security/protected/error.jsp</form-error-page>
```

登录表单的方法必须是 POST，它必须具有一个 j_security_check 的 ACTION 属性、一个名为 j_username 的用户名文本字段以及一个名为 j_password 的口令字段。下面是 Tomcat 安全性示例提供的页面代码，该代码位于%TOMCAT_HOME%\ webapps\jsp-examples\security\protected\login.jsp 文件中，汉化后的代码如下：

```
<html>
<head>
<title>登录</title>
<body bgcolor="white">
<form method="POST" action='<%= response.encodeURL("j_security_check") %>' >
  <table border="0" cellspacing="5">
    <tr>
      <th align="right">用户名:</th>
      <td align="left"><input type="text" name="j_username"></td>
    </tr>
    <tr>
      <th align="right">密码:</th>
      <td align="left"><input type="password" name="j_password"></td>
    </tr>
    <tr>
      <td align="right"><input type="submit" value="登录"></td>
      <td align="left"><input type="reset" value="重置"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

运行该页面，结果如图 10-4 所示。

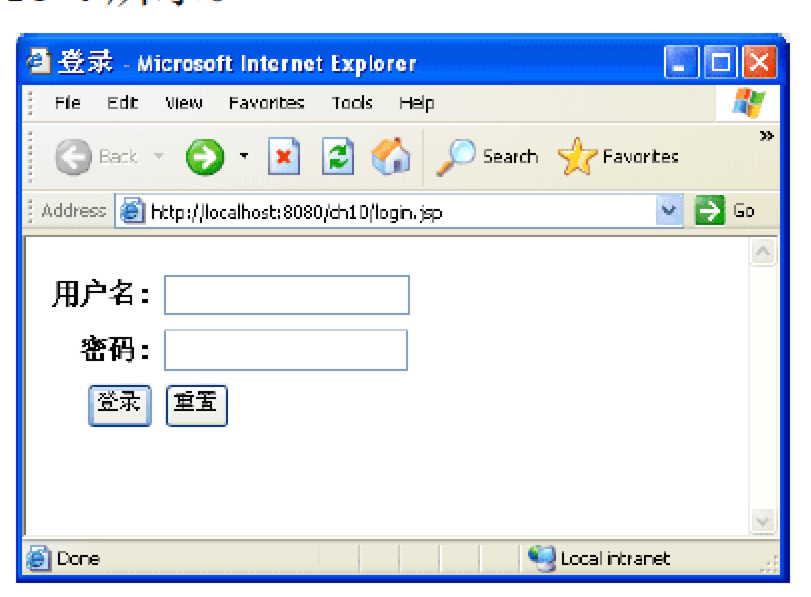


图 10-4　Tomcat 安全性示例的登录页面

Servlet 规范还指定了在使用基于表单的认证时必须发生的操作。下面是当用户请求一个受保护的 Web 资源时发生的操作：(1)容器返回指定的登录表单并存储请求的 URL；(2)客户填写该表单后将表单发送回服务器；(3)容器使用表单域认证用户，如果认证失败则向客户返回指定的错误页面，若认证成功则容器将检查已验证的角色是否可以访问所请求的 URL(Web 资源)；如果不能访问该 Web 资源，则 Servlet 容器返回指定的错误页面，如果能够访问，则将客户重定向到请求的 URL。

下面是 Tomcat 提供的错误页面代码：

```
<html>
<head>
<title>登录失败</title>
</head>
<body bgcolor="white">
错误的用户名/密码，请重新
<a href='<%= response.encodeURL("login.jsp") %>'>登录</a>.
</body>
</html>
```

该页面的运行结果如图 10-5 所示。

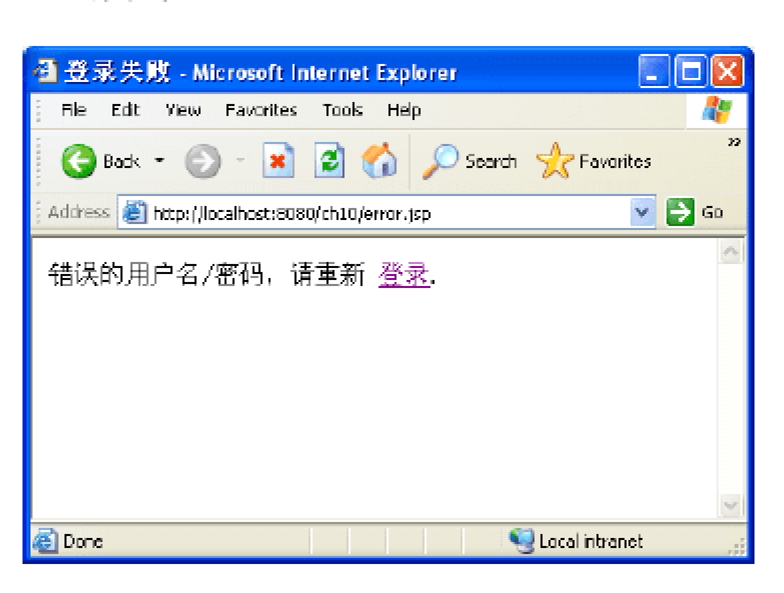


图 10-5　Tomcat 安全性示例的错误页面



10.3.4　HTTPS 客户认证

HTTPS 客户认证是最安全的一种认证形式，因为它要求客户使用数字证书来标识自己。客户认证通常使用 SSL 来实现，一般都被普通的 Web 浏览器所支持。这是一个范围很大的主题域，有兴趣的读者可以在课后查阅相关信息了解更多的相关知识。

10.4　上机练习

本章上机实验主要练习如何使用各种认证方式来保证应用程序的安全性。其中重点掌握如何进行 Web 和 Servlet 容器的认证等操作。

下面以使用 Tomcat 进行 Servlet 容器的基本认证为例进行练习。

(1) 使用资源管理器展开 Tomcat 安装目录，找到并打开 conf 目录。

(2) 使用 XML 编辑器(可以使用记事本)打开 conf 目录下的 tomcat-users.xml 文件。

(3) 在<tomcat-users>模块中加入一个 role 用来测试基本认证的功能，并建立一个 user 对应该 role。如下所示：

```
<tomcat-users>
  <role rolename="testAuth"/>
  <user username="test1" password="password" roles="testAuth"/>
</tomcat-users>。
```

(4) 找到需要测试的应用程序根目录下的 WEB-INF 子目录，用 XML 编辑器打开 web.xml 文件(WEB-INF 目录和 web.xml 见本书第 13 章内容)。假设以 Tomcat 的根路径作为测试，则可以在%Tomcat%/webapps/ROOT 目录下可以找到。

(5) 在 web.xml 中加入<security-constraint>配置模块以确定对哪些资源进行限制以及可以访问的 role 等，如下所示：

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
      <role-name>testAuth</role-name>
  </auth-constraint>
</security-constraint>
```

(6) 在 web.xml 中加入< login-config >配置模块以确定对认证的类型等信息，如下所示：

```
<login-config>
    <auth-method>BASIC</auth-method>
```

```
    <realm-name>Tomcat</realm-name>
  </login-config>
```

(7) 保存所有修改，重新启动 Tomcat 服务，访问 http://localhost:8080/。此时，将弹出一个【输入网络密码】的对话框，只有输入合适的用户名和密码(如 test1/password)后才能正常访问本机的主页。

10.5 习题

10.5.1 填空题



1. 在设计 Web 应用程序时，应该考虑的安全服务有____________、____________、____________、____________、____________和____________。

2. Servlet 2.2 规范要求的认证类型有____________、____________、____________和____________。

10.5.2 选择题

1. 在 Servlet 2.3 规范中，没有被要求的认证类型是(　　　)。

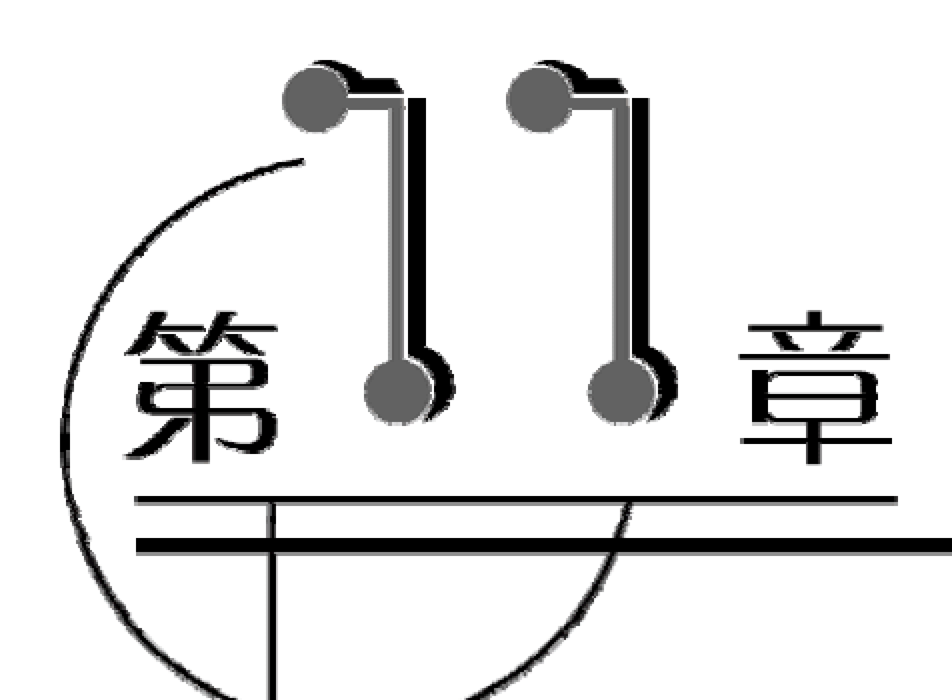
A. 基本认证　B. 摘要认证　C. 基于表单的认证　D. HTTPS 客户认证

2. 以下的安全服务类别中，哪些是设计 Web 应用时必须考虑的(　　)(多选)。

A. 认证　　B. 授权　　C. 审计　　D. HTTPS 客户认证

10.5.3 问答题

1. 什么是认证?
2. 在基于表单的认证中，表单的操作必须是什么?



数据库基础

学习目标

数据库(DataBase)是长期储存在计算机内的、有组织的、可共享的数据集。数据库中的数据按一定的数据模型组织、存储和描述，由数据库管理系统(DataBase Management System)统一管理。DBMS 是一个通用的软件系统，由一组计算机程序构成。它能够对数据库进行有效的管理，并为用户提供了一个软件环境，方便用户使用数据库中的信息。本章就将介绍与数据库技术相关的一些基础知识。

本章重点

- 关系数据库
- SQL 语言
- 数据库对象
- SQL Server 的安装与使用

11.1 数据库基础知识

1963 年，美国 Honeywell 公司的 IDS(Integrated Data Store)系统投入运行，揭开了数据库技术的序幕。 20 世纪 70 年代是数据库蓬勃发展的年代，网状系统和层次系统占据了整个数据库商用市场，而关系系统仅处于实验阶段。20 世纪 80 年代，关系系统由于使用简便以及硬件性能的改善，逐步代替网状系统和层次系统占领了市场。20 世纪 90 年代以来，关系数据库已成为数据库技术的主流，社会各个领域都在广泛使用数据库系统，以提供工作效率，比较流行的关系数据库管理系统有 Oracle、MS SQL Server、DB2、Informix、Sybase、My SQL 以及桌面级数据库 Access 等。

11.1.1 数据库系统使用案例

⊙ 民航机票系统

当人们到民航售票处购买机票时，就需要与数据库打交道。首先工作人员会询问大家需要什么时间的机票，然后根据这个时间提交一个相应的查询命令给民航机票系统，系统处理完后返回结果，工作人员依据查询结果就可以告知购票者该时间是否有票？有无头等舱？经济舱的打折情况等信息。当决定购买机票后，工作人员会要求大家提供正确的身份信息，以便录入系统之中，同时提交购票请求，请求被成功接受后，系统将对数据库中该航班的票数做减 1 操作，并进行出票操作。

⊙ 药店管理信息系统

现在大一点的药店基本都配备了药品信息管理软件，当人们进到一家药店买药，在付钱时会发现营业员会将该药品信息扫描入计算机系统，由相应的药品信息管理软件做相应处理。一般地，药店对于药品信息管理系统会有如下需求。

(1) 录入功能：当药店进药时，提供录入药品信息的功能。相关信息有批次号、药品代号、药品名称、生产厂家、有效期、库存量。可建立表 medicine 存放以上信息。

(2) 售药功能：出售某药品时，能够根据数量相应修改该药品的库存量。建议尽量先卖有效期限短的批次药品。

(3) 查询功能：能够查询某药品各批次的总库存量以及预警即将(如半年内)要过期的药品等信息。

(4) 数据清理：将库存量为 0 的药品条目删除掉。

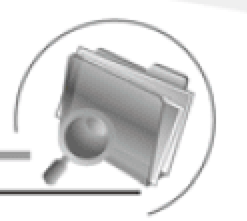
⊙ Web 教务网站

现在人们已不满足 Web 技术发展初期由文本、图形和超链接组成的静态信息发布与浏览，而是需要将 Web 技术与数据库技术集成在一起，客户端通过 Web 页面与服务器进行信息的交互及传递，并通过 Web 页面对后台数据库进行远程管理和控制。Web 教务系统使前台页面具备动态的发布新闻、通知、公告等功能，并能够提供管理文件供师生下载，这样可以使得教务处的信息发布与管理，从先前纸质公文的形式转换到即时、形象、高效的电子页面形式，极大的推动学校教务管理工作效率的提高。本书将在第 15 章详细介绍如何构建基于数据库的 JSP 动态教务网站。

11.1.2 数据库基本概念

⊙ 信息、数据与数据处理

数据是记载信息的各种物理符号，它是信息的载体、信息的具体表现形式。在日常生活中，数据无所不在。数字、文字、图像、图表、声音等都是数据，人们通过数据来认识世界，交流信息。数据是数据库管理的基本内容和对象。



数据与信息是密切联系的。信息是向人们提供关于现实事物的知识；数据则是表示信息的物理符号，二者是不可分离而又有一定区别的两个相关概念。

数据处理是指对数据进行一系列收集、加工、储存、合并、分类、计算、检索、传输等操作过程。在当今的信息社会，通常所说的信息处理实际上就是指利用计算机进行数据处理的过程。另外，数据处理又是与数据管理相联系的，数据管理技术的好坏，将直接影响数据处理的效率。一般认为，数据处理经历了这么 4 个不同的阶段。

(1) 人工处理阶段：该阶段还没有出现操作系统软件。

(2) 文件系统阶段：文件系统包含在操作系统内。

(3) 数据库系统阶段：出现了专门进行数据处理的 DBMS。

(4) 高级数据库阶段：出现了分布式数据库、面向对象数据库以及数据仓库等。

⦿ 数据库、数据库管理系统与数据库系统

数据库(Database)是指按一定数据结构组织并存储在计算机中的一组相关数据的集合。它能为各种用户共享，具有最小的冗余；数据之间密切联系，而又有较高的独立性。

数据库管理系统(DBMS)可以理解为管理数据库的系统软件。实际上 DBMS 是用户与操作系统之间的一层管理软件，它为用户或应用程序提供访问数据库的方法，包括数据库的创建、维护、管理、备份、恢复、数据复制等工作。

数据库系统(Database System，简称 DBS)是指在计算机系统中引入数据库后构成的系统，一般由数据库、数据库管理系统(及其开发工具)、应用系统、数据库管理员和用户组成。

11.1.3　实体以及数据模型

⦿ 实体

客观存在，可以相互区别的事物称为实体。现实世界中的客观事物基本都可以称之为实体，它可以指人，如一个教师，一个学生等；也可以指物，如一本教材，一间教室等。它不仅可以指实际的物体，而且可以指抽象的事件，如一门课程的讲授，一场明星演出等。它还可以指事物与事物之间的联系，如学生选课，客户订货等。

⦿ 实体间的联系

实体之间的对应关系称为联系，它反映了现实世界的事物之间的相互关联。例如，图书和出版社之间的关联关系就是：一个出版社可出版多种书，同一种书只能在一个出版社出版。实体之间的联系是指一个实体集中可能出现的每一个实体与另一实体集中多少个具体实体存在联系。实体之间有各种各样的联系，归纳起来主要有 3 种类型。

一对一联系(1：1)：如果对于实体集 A 中的每一个实体，实体集 B 中有且只有一个实体与之联系。反之亦然，则称实体集 A 与实体集 B 具有一对一的联系。

一对多联系(1：n)：如果对于实体集 A 中的每一个实体，实体集 B 中有多个实体与之联系，反之对于实体集 B 中的每一个实体，实体集 A 中至多只有一个实体与之联系，则称实体集 A 与实体集 B 有一对多的联系。



多对多联系(m: n)：如果对于实体集 A 中的每一个实体，实体集 B 中有多个实体与之联系，而对于实体集 B 中的每一个实体，实体集 A 中也有多个实体与之联系，则称实体集 A 与实体集 B 之间有多对多的联系。

⊙ 实体属性

一个实体可以有不同的属性，属性描述了实体在某一方面的特性。例如，教师实体可以用教师编号、姓名、性别、出生日期、职称、基本工资、研究方向等属性来描述。每个属性可以取不同的值，对于具体的某一教师，其编号为 140011，姓名为张三，性别为男，出生日期为 1973 年 6 月 7 日，职称为副教授，基本工资为 1078 元，研究方向为分布式数据库技术，分别为上述教师实体属性的取值。属性值的变化范围称作属性值的域，如性别这个属性的域为(男，女)，职称的域为(助教，讲师，副教授，教授)等。由此可见，属性是个变量，属性值是变量所取的值，而域是变量的取值范围。

由上可见，属性值所组成的集合表征一个实体，相应的属性的集合表征了一种实体的类型，称为实体型。例如上面的教师编号、姓名、性别、出生日期、职称、基本工资、研究方向等表征【教师】这一实体的实体型。同类型的实体的集合称为实体集。

用【表】来表示同一类实体，即实体集，用【记录】来表示一个具体的实体，用【字段】来表示实体的属性。显然，字段的集合组成一条记录，记录的集合组成一个表，相应于实体型，则代表了表的结构。

⊙ 数据模型

数据模型是数据库的组织形式，是实体模型的数据化。每一个实体的数据称为【记录】；实体属性的数据称为【数据项】或【字段】；所有记录的集合组成【表】。目前，数据库系统中主要的数据模型有 3 种：层次模型、网状模型和关系模型。人们最常用的是关系型数据库，如 Oracle 和 SQL Server 都属于关系型数据库。

11.1.4 关系数据库

⊙ 关系模型

关系模型把数据之间的组织关系用一张表来表示，它的数据结构是一个二维表，一个二维表就称为一个关系。二维表中的一行称为【记录】，表中的属性(数据项)称为列(也就是字段)。在数据库中一张二维表就构成了一个数据库文件，反之一个数据库文件就对应着一张二维表。

二维表构成的关系模型应该满足以下条件：表格中不允许有重复的行和列；表格中的各列不允许有相同的列名；表格中每一列的所有数据类型必须相同或兼容。

关系模型的优点：数据结构单一，在关系模型中，不管是实体还是实体之间的联系，都用关系来表示，而关系都对应着一张二维数据表，数据结构简单，清晰；关系规范化，并建立在严格的理论基础上，关系中的每个属性不可再分割，关系是建立在严格的数学概念基础上，具有坚实的理论基础；概念简单，操作方便，关系模型最大的优点就是简单，用户容易理解和掌握，一个关系就是一张二维表，用户只需用简单的查询语言就能对数据库进行操作。



◉ 关系型数据库

以关系模型建立的数据库就是关系型数据库(RDB:Relational Database)。关系数据库中包含若干个关系，每个关系都由关系模式确定，每个关系模式包含若干个属性和属性对应的域，所以，定义关系数据库就是逐一定义关系模式，对每一个关系模式逐一定义属性及其对应的域。一个关系就是一张二维表格，表格由表格结构与数据构成，表格的结构对应关系模式，表格中的每一列对应关系模式的一个属性，该列的数据类型和取值范围就是该属性的域。因此，定义了表格就定义了对应的关系。

11.2　结构化查询语言 SQL

SQL(Structure Query Language)即结构化查询语言，是标准的数据库查询语言，它是一种面向数据库的通用数据处理语言规范。SQL 语言的主要功能是同各种数据库系统建立联系，进行沟通。按照 ANSI(美国国家标准协会)的规定，SQL 被作为关系型数据库管理系统的标准语言。SQL 语句可以用来执行各种各样的操作，例如更新数据库中的数据、从数据库中提取数据等。目前，绝大多数流行的关系型数据库管理系统，如 Oracle、Sybase、 Microsoft SQL Server、Access 等都采用了 SQL 语言标准。虽然很多数据库都对 SQL 语句进行了再开发和扩展，但是包括 Select、Insert、Update、Delete、Create 以及 Drop 在内的标准 SQL 命令对于几乎所有的数据库管理系统来说，则是通用的。



11.2.1　SQL 的语言元素

SQL 语言由 3 种元素构成：数据定义语言(DDL)、数据操纵语言(DML)和数据控制语言(DCL)，它们均不区分大小写。

1. 数据定义语言(DDL)

数据定义语言用于创建数据库、定义 SQL 模式、基本表、视图和索引。包括 create、drop、alter 等语句。这 3 个命令用于创建、删除、更改一个对象，它们是影响数据库结构的语句。

假如需要创建名为 company 的数据库，并且该数据库中有两个基本表：employee(职员表)和 department(部门表)，那么可以采用以下的 SQL 语句来创建。

```
  创建数据库
create database company;
  创建员工基本信息表
  create table employee(
      empno       varchar(8) primary key,          工号
      empname     varchar(10),                     员工姓名
      deptno      varchar(2),                      部门号
```

```
        m_salary int,                          月薪
        job         varchar(20),               工种
        commission   int                       佣金
)
   创建部门基本信息表
   create table department(
        deptno     varchar(2)   primary key,   部门号
        location varchar(10)                   部门所在地
)
```

2. 数据操纵语言(DML)

数据操纵语言分为数据查询和数据更新两类。其中数据更新又分为插入、删除和修改 3 种。

例如：insert、select、update、delete 语句。这 4 个命令分别用于查询、插入、修改和删除数据。



下面的语句表示查询公司所有员工信息：

```
select * from employee
```


3. 数据控制语言(DCL)

这一部分包含对表和视图等对象的授权、完整性规则的描述以及事务控制等。它主要用于改变与某个数据库用户或角色相关联的权限。

例如：grant、deny、revoke 语句。grant 允许一个用户能够访问数据库或运行某些 SQL 语句。deny 用于剥夺某个安全账号的访问权限，并阻止某个用户、用户组或角色从他的组和角色成员中继承原有的权限。revoke 用于删除一个以前授予或拒绝的权限。默认情况下只有 sysadmin、dbcreator 以及 db_Owner 等用户才能执行这些语句。

下面主要讲述数据操纵语言(DML)，因为这些是最常用到的 SQL 语句。

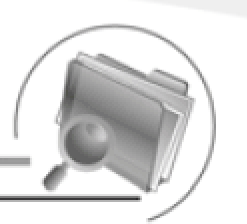
11.2.2 INSERT 语句

INSERT 语句的作用是添加一条或多条记录到数据表中，因此数据库的录入功能可以通过它来实现。

- 插入子查询结果

子查询不仅可以嵌套在 SELECT 语句中，用以构造父查询的条件，也可以嵌套在 INSERT 语句中，用以生成要插入的数据。其功能是批量插入，一次将子查询的结果全部插入指定表中。其基本语法格式如下：

```
INSERT   INTO <表名> [(<属性列 1> [，<属性列 2>…]) 子查询
```



【例 11-1】插入子查询结果。

employee_与 employee 表结构一样，要求将前表中的销售员名单插入到后表中(假定无冲突)

```
INSERT INTO employee
SELECT *   FROM employee_
WHERE job = 'salesman'
```

◉ 插入单条记录

其基本语法格式如下：

```
INSERT INTO <表名> [(<属性列 1>[，  <属性列 2> ...])]
    VALUES (<常量 1>[，  <常量 2>[，  ...]])
```

【例 11-2】插入单条记录。

录入员工张三的基本信息：工号 1、部门号 20、月薪 3300、工种为销售员且佣金为 3000。

```
INSERT INTO employee (empno,empname,deptno,m_salary,job,commission)
VALUES('1','张三','20',3300,'salesman',3000)
```

录入 20 号部门的信息：所在地为北京。

```
INSERT INTO department (deptno,location) values ('20','北京');
```



11.2.3　SELECT 语句

SQL 语言的核心语句就是数据库查询语句 SELECT，它是所有 SQL 语句中使用最频繁的一个，也是较复杂的一个。

◉ 基本语法

Select 语句用于从数据库中检索行。一个最简单的 Select 语法形式如下：

```
SELECT field1[，field2…]
FROM table1[，table2…]
```

SELECT 和 FROM 是关键字。FROM 指定从数据库中取出数据的表，SELECT 指定从表中取出哪些字段值(即列)。如下面的示例，从 employee 表中取出 empname，m_salary 和 job 字段：

```
SELECT empname，  m_salary，  job
FROM employee
```

◉ 从表中取出所有字段值

如果要从表中取出所有字段值，可以使用符号*来代替所有字段名，如下面的语句：

```
SELECT *
FROM employee
```

⊙ 使用关键字 AS 为字段重新指定列名

使用 SELECT 语句选择记录时，可以用 AS 关键字为字段指定一个新的列名。使用以下语句可以将 empname，m_salary 和 job 字段名分别显示为【员工姓名】、【月薪】和【工种】：

```
SELECT empname as  员工姓名，m_salary as  月薪，job as  工种
FROM employee
```

⊙ 使用 WHERE 关键字查询符合一定条件的记录

以上所有的 SELECT 语句都是从表中取出所有的记录，也可以在 SELECT 语句中使用 WHERE 关键字，查询符合一定条件的记录。如用以下语句，就可以只显示月薪超过 3000 的员工记录：




```
SELECT empname， m_salary， job
FROM employee
WHERE m_salary > 3000
```

其中 m_salary > 3000 为条件表达式，在 WHERE 子名中可以使用不同形式的条件表达式。

SQL 语句中常用的算术运算符有：+(加)、-(减)、*(乘)、\(除)和%(模)；常用的比较运算符有：=(等于)、>(大于)、<(小于)、>=(大于或等于)、<=(小于或等于)和！=(不等于)；常用的逻辑运算符有：AND(与)、OR(或)、NOT(取反)和 LIKE(模糊匹配)。

需要注意的是 LIKE 运算符的用法：该运算符用于确定给定的字符串是否与指定的模式匹配。模式可以包含常规字符和通配符字符，在模式匹配过程中常规字符必须与字符串中指定的字符完全匹配。可以使用通配符与字符串中的任意部分匹配，这样使 LIKE 运算符更加灵活。

在 SQL 语句中可以使用的通配符有以下这些。

%：用于代替零个或多个字符的任意字符串。

_(下划线)：用于代替单个字符。

[]：用于指定一定范围的字符。

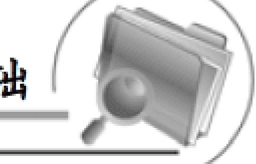
[^]：用于指定不属于某一范围的字符。

【例 11-3】使用 WHERE 关键字查询符合一定条件的记录。

```
SELECT empname， m_salary， job
FROM employee
WHERE empname LIKE '张%'
```

该示例使用通配符%，结果将显示 employee 表中姓张的所有员工记录。

```
SELECT empname， m_salary， job
FROM employee
WHERE empname LIKE '张_'
```



该示例使用通配符_，结果将显示 employee 表中姓张且名字为两个字的所有员工记录。

```
SELECT empname， m_salary， job
FROM employee
WHERE empname LIKE 'mi[a-h]e'
```

该示例使用通配符[]，结果将显示 employee 表中 empname 字段值长度为 4 个字母并且以 mi 开头，e 结尾，第 3 个字母在 a 与 h 之间的所有员工记录。

```
SELECT empname， m_salary， job
FROM employee
WHERE empname LIKE '[^a-h]e'
```

该示例使用通配符[^]，结果将显示 employee 表中 empname 字段值长度为 2 个字母并且第一个字母不包含 a 到 h 的字母的所有员工记录。

下面再列举几个带有 where 条件子句的例子。

【例 11-4】where 条件子句。

查询没有佣金的员工：

```
SELECT empname， m_salary， job
FROM employee
WHERE commission is null
```

查询既不是销售员也不是经理的员工姓名，月薪及工种：

```
SELECT empname， m_salary， job
FROM employee
WHERE job not in ('salesman'，'manager')
```

查询出员工张三的工作地点：

```
SELECT a.location
FROM department a , employee b
WHERE a.deptno = b.deptno and b.empname = '张三'
```

查询月薪比张三高的员工：

```
SELECT empname
FROM employee
WHERE m_salary > (SELECT m_salary FROM employee WHERE empname = '张三' )
```

上面这个 SELECT 语句的 WHERE 子句中又使用了一个 SELECT 语句，人们把这种情况称之为嵌套。嵌套 SELECT 语句常可以用来解决复杂的查询任务。

◉ 将多列内容合并到一列中

可以使用 || 将多个字段的内容合并到一列中，如下面的示例：

```
SELECT deptno || empno    AS emp_no
FROM employee
```

◉ 使用 ORDER BY 关键字对结果进行排序

使用 ORDER BY 关键字可以对查询结果进行排序，ORDER BY 后面可以有多个字段名，按照出现的顺序依次排序。

【例 11-5】使用 ORDER BY 关键字对结果进行排序。

```
SELECT empname， m_salary， job
FROM employee
ORDER BY m_salary
```



该语句将结果按照 m_salary 列的值从小到大排序。可以指定排列顺序是按升序(ASC)或者降序(DESC)排列，默认的排列顺序是升序(ASC)。

◉ 使用 DISTINCT 关键字

使用 DISTINCT 关键字可以消除返回结果中的重复项。

【例 11-6】使用 DISTINCT 关键字。

查询公司员工的工种有哪几种

```
SELECT DISTINCT job
FROM employee
```

该语句取出 employee 表中的不同 job 值，也就是说如果表中有多个相同的 job 值，则只取其中的一个。

11.2.4 UPDATE 语句

UPDATE 语句的作用是修改表中已有的一条或多条记录。其基本语法格式如下：

```
UPDATE <表名>
  SET <列名>=<表达式>[，<列名>=<表达式>]…
  [WHERE <条件>]
```

UPDATE 关键字用于指定修改哪个表；SET 关键字指定修改哪些字段以及修改后的值，这些字段必须是表中存在的字段；WHERE 关键字用于限定修改符合特定条件的记录，如果省略 WHERE 子句，则表示要修改表中的所有记录。

【例 11-7】UPDATE 语句。

员工张三被调往 10 号部门，同时为其加薪 20%：



```
update employee
set deptno='10' , m_salary= m_salary*1.2
where empname='张三'
```

公司 20 号部门被迁往广州：

```
update department
set location='广州'
where deptno='20';
```

11.2.5　DELETE 语句

DELETE 语句用于从表中删除一条或多条记录。其基本语法格式如下：

```
DELETE
    [FROM] <表名>
[WHERE <条件>]
```

DELETE 关键字指定需要删除记录的表名；WHERE 关键字用于限定删除符合特定条件的记录，如果省略，则删除整个表中的记录。

【例 11-8】DELETE 语句。

从 employee 表中删除所有员工记录，语句如下所示。

```
DELETE FROM employee
```

删除所有在 10 号部门工作的员工信息，语句如下所示。

```
DELETE
FROM employee
WHERE deptno = '10'
```

11.3　数据库对象

数据库对象包括表、视图、索引、存储过程等，下面简单介绍其中比较重要的几个对象。

11.3.1　表

表是包含数据库中所有数据的数据库对象。表定义为列的集合，与电子表格相似，数据在表中是按行和列的格式组织排列的。表中的每行代表惟一的一条记录，而每列则代表记录中的



一个域。例如，在包含公司雇员数据的表中每一行代表一名雇员，各列分别表示雇员的详细资料，如雇员编号、姓名、地址、职位以及电话号码等。

1. 创建表

创建表的基本语法格式如下：

```
CREATE TABLE <表名>(<列名><数据类型> [列级完整性约束条件] [，<列名> <数据类型> [列级完整性约束条件]...] [，<表级完整性约束条件>])
```

在前面已经创建过公司的员工表以及部门表，下面用 CREATE 语句再创建一个关于雇员的表 employee：

```
CREATE TABLE employee
 (
        emp_id    number         CONSTRAINT PK_emp_id PRIMARY KEY ，
        fname     varchar(20)    NOT NULL，
        lname     varchar(30)    NOT NULL，
        job_id    smallint       NOT NULL DEFAULT 1 REFERENCES jobs(job_id)，
        job_lvl tinyint          DEFAULT 10，
        hire_date datetime       NOT NULL DEFAULT (getdate())
 )
```

2. 修改表

表创建之后可以修改许多已定义的选项，包括：添加、修改、删除列，例如，列的名称、长度、数据类型、精度、小数位数以及为空性均可进行修改，不过有一些限制而已；还可以添加或删除 PRIMARY KEY 和 FOREIGN KEY 约束；添加或删除 UNIQUE 和 CHECK 约束及 DEFAULT 定义(对象)等。

其基本语法格式如下：

```
ALTER TABLE <表名>[ADD <新列名><数据类型>[完整性约束]][DROP COLUMN<列名><完整性约束名><完整性约束名>] [MODIFY<列名> <数据类型><数据类型>]
```

其中<表名>指定需要修改的表，ADD 子句用于增加新列和新的完整性约束条件，DROP 子句用于删除指定列和指定的完整性约束条件，MODIFY 子句用于修改原有的列定义。

下面看几个例子。

【例 11-9】更改表以添加新列。

添加一个允许空值的列，而且没有通过 DEFAULT 定义默认值，各行的新列中的值将为 NULL。

```
CREATE TABLE employee1 ( column_a INT)
GO
ALTER TABLE employee1 ADD column_b VARCHAR(20) NULL
GO
```



【例 11-10】更改表以添加具有约束的列。

向表中添加具有 UNIQUE 约束的新列。

```
CREATE TABLE employee2( column_a INT)
GO
ALTER TABLE employee2 ADD column_b VARCHAR(20) NULL
    CONSTRAINT exb_unique UNIQUE
GO
```

【例 11-11】更改表以删除列。

修改表以删除一列。

```
CREATE TABLE employee3 (column_a INT，  column_b VARCHAR (20) NULL)
GO
ALTER TABLE employee3 DROP COLUMN column_b
GO
```



3. 删除表

有时需要删除已有的表(如当要实现新的设计或释放数据库空间时)。删除表时，表的结构定义、数据、全文索引、约束和索引都将永久地从数据库中删除，原来存放表及其索引的存储空间可以用来存放其他表。如果不想等待临时表自动除去，可以明确删除临时表。

其基本语法格式如下：

```
DROP TABLE <表名>
```

基本表定义一旦删除，表中的数据、在此表上建立的索引都将自动被删除，而建立在此表上的视图虽然仍保留，但已无法引用。因此执行表删除操作时一定要格外小心。

11.3.2　索引

数据库中的索引与书籍中的索引类似。在一本书中，利用索引可以快速查找所需信息，而无须阅读整本书。在数据库中，索引使得数据库程序无须对整个表进行扫描，就可以在其中找到所需数据。书中的索引是一个词语列表，其中注明了包含各个词的页码。而数据库中的索引是一个表中所包含的值的列表，其中注明了表中包含各个值的行所在的存储位置。可以为表中的单个列建立索引，也可以为一组列建立索引；索引采用 B 树结构。索引包含一个条目，该条目有来自表中每一行的一个或多个列(搜索关键字)。B 树按照搜索关键字排序，可以在搜索关键字的任何子词条集合上进行高效搜索。例如，对于一个 A、B、C 列上的索引，可以在 A 以及 A、B 和 A、B、C 上对其进行高效的搜索。

大多数书都包含一个关于词汇、名称、地点等的总索引。数据库则包含分别关于所选类型或数据列的索引：这好比在一本书中分别为人名和地名建立索引。当创建数据库并优化其性能

时，应该为数据查询所使用的列创建索引。

1. 创建索引

其基本语法格式如下：

```
CREATE [UNIQUE] INDEX 索引名
ON 基本表名(列名[，列名…n])
```

【例 11-12】创建索引。

给员工表的员工号字段创建索引，以提高按工号查询员工信息的速度。

```
CREATE INDEX index_01
ON employee(empno)
```

2. 删除索引

其基本语法格式如下：

```
DROP INDEX 索引名
```

例如：

```
    DROP INDEX index_01
```

11.3.3 视图

视图是一个虚表，其内容由查询来动态定义。同真实的基表一样，视图包含一系列带有名称的列和行记录。但是，视图并不在数据库中以存储的数据值集的形式存在，其数据完全来自于定义视图的查询所检索的表，并且在引用视图时动态生成。数据库内存储的是视图的定义 SELECT 语句，SELECT 语句的执行结果集构成视图所返回的虚表。

使用视图可以实现下列任意一个或所有功能：将用户限定在表中的特定行上，例如，只允许雇员看见工作跟踪表内记录其工作的行；将用户限定在特定列上，例如，对于那些不负责处理工资单的雇员，只允许他们看见雇员表中的姓名列、办公室列、工作电话列和部门列，而不能看见任何包含工资信息或个人信息的列；将多个表中的列连接起来，使它们看起来像一个表；聚合信息而非提供详细信息，例如，显示一个列的和或列的最大值和最小值。

下面介绍一些基本的视图操作。

1. 创建视图

其基本语法格式如下：

```
CREATE VIEW <视图名>[(<列名>[，<列名>]…)] AS <子查询>
```



其中子查询可以是任意复杂的 SELECT 语句，但通常不允许含有 ORDER BY 子句和 DISTINCT 短语。

通过定义 SELECT 语句检索出将在视图中显示的数据来创建视图。SELECT 语句引用的数据表称为视图的基表。在下例中， empeview 是一个视图，该视图选择 2 个基表(employee 和 department)中的数据来构成包含员工工作地点信息的虚表：

```
CREATE VIEW empview
AS
SELECT empname， m_salary， job， location
FROM employee,department
WHERE employee.deptno=department.deptno
```

视图创建之后，就可以用引用表时所使用的方法引用 empview。

```
SELECT *
FROM empview
```



一个视图也可以引用另一个视图。例如，empview 显示的信息对公司管理人员很有用，但公司对于个人薪水通常是采取保密方式的，因此可以建立一个新视图，在其中包含除 m_salary 之外的所有 empview 列。使用这个新视图，公司可以将信息发布出来给所有员工，方便员工间的联系交流。

```
CREATE VIEW allview
AS
SELECT empname， job， location
FROM empview
```

2. 修改视图

在完成视图定义之后，可以在不删除和重新创建视图的条件下更改视图名称或修改其定义，从而丢失与之相关联的权限。在重命名视图时，要遵循以下原则：要重命名的视图必须位于当前数据库中；新名称必须遵守标识符规则；只能重命名自己拥有的视图；数据库所有者可以更改任何用户视图的名称。

下面创建一个名为 All_employee 的视图，该视图包含公司全部员工，并将其 SELECT 权限授予了 public 用户。而后假定需要修改视图，使其仅包含工种为销售员的公司员工信息，于是，采用 ALTER VIEW 语句对该视图进行修改。

```
CREATE VIEW All_employee (empno， empname， job， m_salary， commission)
AS
SELECT empno， empname， job， m_salary， commission
FROM employee
GO
GRANT SELECT ON All_employee TO public
```




```
GO
ALTER VIEW All_employee (empno， empname， job， m_salary， commission)
AS
SELECT empno， empname， job， m_salary， commission
FROM employee
WHERE job = 'salesman'
GO
```

3. 删除视图

视图创建后，如果不再需要该视图，或想清除视图定义以及与之相关联的权限，可以删除该视图。删除视图后，表和视图所基于的数据并不会受到影响。任何使用基于已删除视图的查询将会失败，除非创建了同名称的另一个视图。

删除视图基本语法格式如下：



```
DROP VIEW <视图名>
```

一旦视图被删除后，由该视图导出的其他视图也将失效，用户应该使用 DROP VIEW 语句将他们逐一删除。

4. 通过视图更新基表

通过视图可以实现往基表中插入、修改和删除数据。在视图中添加数据，就是对视图的基表添加数据，添加数据也使用 INSERT 命令。用法与向基表中添加数据相同。但要注意：

(1) 当视图有多个基表时，不能在一个语句中对视图中的多个基表同时进行修改，亦即一次只能对视图的一个基表内的数据进行增、删、改操作。

(2) 由于视图中可能仅仅包含基表中的部分列，因此通过视图向基表中插入数据时，要求基表中的其他非视图列应该是允许空(NULL)或含有默认值。

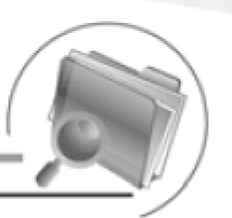
(3) 插入的数据必须满足基表的约束条件。

同样，使用 UPDATE 命令可以通过视图修改基表中的数据，修改数据的方法与修改表中数据的方法相同。与通过视图向基表插入数据一样，也要注意一次只能修改一个表，且修改后的数据不得违反基表中的各种约束关系。

另外，在视图中使用 DELETE 命令删除数据就是删除视图基表中的对应的数据。语法与在表中删除数据相同。在视图中删除数据得以进行的条件是视图只能有一个基表，且所要删除的数据必须存在于视图中(即不违反视图的建立条件)。删除视图数据是针对视图的行进行的，亦即 DELETE 命令将删除视图一行中的所有数据，实际上是删除了基表中对应于该视图中某一行(或几行)的数据。

11.3.4 存储过程

存储过程是由流控制和 SQL 语句书写的过程，该过程经编译和优化后存储在数据库服务器



中，使用时只要调用即可。在 SQL Server 中，若干个有联系的过程可以组合在一起构成程序包。

使用存储过程有以下优点：存储过程的能力大大增强了 SQL 语言的功能和灵活性，它可以用流控制语句编写，有很强的灵活性，可以完成复杂的判断和较复杂的运算；可以保证数据的安全性和完整性；通过存储过程可以使没有权限的用户在控制之下间接地存取数据库，从而保证数据的安全；通过存储过程可以使相关的动作在一起发生，从而可以维护数据库的完整性；在运行存储过程之前，数据库已经对其进行了语法和句法分析，并给出了优化执行方案，这种已经编译好的过程可以极大地改善 SQL 语句的性能，由于执行 SQL 语句的大部分工作已经完成，所以存储过程能以极快的速度执行；可以降低网络的通信量。

不同数据库系统存储过程的写法不一，下面以 SQL Server 为例介绍存储过程的用法。

SQL Server 创建存储过程的语法格式如下：

```
CREATE PROCEDURE [owner].procedure_name
[({@parameter data_type}[VARYING][=default][OUTPUT])]
[，...n ]
[WITH {RECOMPILE|ENCRYPTION|RECOMPILE，ENCRYPTION}]
AS
Sql_statements
```

该语法中的参数说明如下。

- owner：拥有存储过程的用户 ID 的名称，必须是当前用户的名称或当前用户所属的角色的名称。
- procedure_name：新存储过程的名称。过程名必须符合标识符规则，且对数据库及其所有者必须惟一。
- @parameter：在存储过程中包含的输入和输出参数。
- data_type：参数的数据类型。
- VARYING：指定作为输出参数支持的结果集(由存储过程动态构造，内容可以变化)，仅适用于游标参数。
- default：为这个参数指定的默认值，必须是一个常量。如果该参数使用在 like 子句中，则该默认值可以含有通配符。如果给出了默认值，那么在执行该存储过程时，如果没有向具有默认值的参数传递参数值时，则使用该默认值。
- OUTPUT：表明参数是返回参数。
- {RECOMPILE | ENCRYPTION | RECOMPILE， ENCRYPTION}：RECOMPILE 表明 SQL Server 不会缓存该过程的计划，该过程将在运行时重新编译；ENCRYPTION 表示 SQL Server 加密 syscomments 表中包含 CREATE PROCEDURE 语句文本的条目。
- sql_statements：在存储过程中要执行的动作。

可以建立一个简单存储过程，当用户修改数据库的重要数据时，即把用户的用户名、修改日期，修改操作记录到一个日志表中：

```
CREATE PROCEDURE update_log AS
  BEGIN
```

```
        insert into update_log_tab(use_name，update_date，operation)
          values(user，sysdate，'update')
    END
  GO
```

可以在恰当的位置调用这个存储过程来记录用户对表的修改。例如：在表 sal_comm 上创建一个修改触发器，当用户修改此表后，用户的名称、修改时间和操作即被记录在了表 update_log_tab 中：

```
CREATE TRIGGER audit_update
ON sal_comm
AFTER UPDATE
AS
  EXEC update_log
GO
```


11.4 SQL 的统计函数

在 SELECT 语句中除了可以使用算术表达式进行列计算外，还可以使用一系列的统计函数对表中的所有数据进行汇总、统计等多种运算，统计函数属于聚合函数(Aggregate Function)。统计函数通常用于 SELECT 语句中，作为结果集中的返回列。

11.4.1 SUM 函数

SUM 函数返回表达式中所有值的总和，其语法格式如下：

```
SUM([ ALL | DISTINCT ] expression)
```

【例 11-13】下面的语句表示对 employee 表中的 m_salary 字段求和，即统计公司一个月需支付的工资总额。

```
SELECT SUM(m_salary)
FROM employee
```

11.4.2 AVG 函数

AVG 函数返回表达式中所有值的平均值，其语法格式如下：

```
AVG([ ALL | DISTINCT ] expression)
```



参数说明如下。

- ALL：对所有的值进行计算。ALL 是默认值。
- DISTINCT：如果一个值出现了多次，则只取其中的一个进行计算。
- expression：数值表达式(bit 数据类型除外)，也可以是字段名(字段的类型必须是数值类型)。不允许使用聚合函数和子查询。

【例 11-14】AVG 函数。

下面的语句表示对 employee 表中的 m_salary 字段求平均值。

```
SELECT AVG(m_salary)
FROM employee
```

下面的语句表示统计公司各个工种员工的平均月薪。

```
SELECT job,AVG(m_salary)
FROM employee
GROUP BY job
```

下面的语句表示查询薪水超过其所在部门平均薪水的所有员工。

```
select a1.empname
from employee a1
where m_salary > (select    avg(m_salary) from employee a2
                            where a1.deptno = a2.deptno
                            )
```



11.4.3　COUNT 函数

COUNT 函数用于返回表达式中项目的数量，其语法格式如下：

```
COUNT([ ALL | DISTINCT ] expression | *)
```

参数说明如下。

- ALL：对所有的非空值进行计算，ALL 是默认值。
- DISTINCT：如果一个值出现了多次，则只取其中的一个进行计算，并且只对非空值进行计算。
- expression：一个表达式。其类型可以是除 uniqueidentifier、text、image 或 ntext 之外的任何类型。不允许使用聚合函数和子查询。
- *：计算表中所有的行已返回表中记录的总数，包括含有空值的行。

【例 11-15】COUNT 函数。

下面的语句将求出 employee 表中所包含的记录数，即公司员工总数。

```
SELECT count(*)
FROM employee
```

统计公司各个部门的员工数。

```
SELECT deptno,count(*)
FROM employee
GROUP BY deptno
```

11.4.4 Min 和 Max 函数

MIN 函数返回表达式中的最小值，而 MAX 返回表达式中的最大值，其语法格式如下：

```
MIN([ ALL | DISTINCT ] expression)
MAX([ ALL | DISTINCT ] expression)
```


参数 ALL 和 DISTINCT 与 AVG 和 SUM 函数中的相应参数基本相同，expression 是常量、列、函数或者算术、按位与字符串等运算符的任意组合。MAX 可以使用数字列、字符列和 datetime 列，但不能用于 bit 列，也不允许使用聚合函数和子查询。

【例 11-16】Min 和 Max 函数。

下面的语句将返回 employee 表中 m_salary 列中的最大值。

```
SELECT MAX(m_salary)
FROM employee
```

查询公司各个部门的最高薪水和最低薪水，要求最高薪水比最低薪水超出 1000。

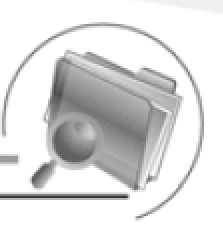
```
SELECT deptno, MAX(m_salary),MIN(m_salary)
FROM employee
GROUP BY deptno
Having MAX(m_salary) - MIN (m_salary) > 1000
```

11.5 SQL Server 简介

本节将介绍 SQL Server 2000 的安装及企业管理器的一些基本操作。

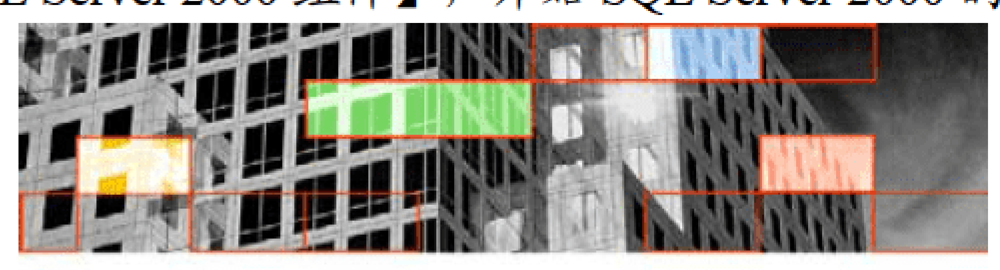
11.5.1 安装 SQL Server 2000

要进行数据库应用的开发，首先需要在本机安装 SQL Server 2000 Developer Edition。SQL Server 的安装相对比较复杂，需要进行一定的配置。下面简要介绍其安装过程。



(1) 把安装光盘放入光驱中，SQL Server 的安装程序会自动启动，如图 11-1 所示。

(2) 单击【安装 SQL Server 2000 组件】，开始 SQL Server 2000 的安装，如图 11-2 所示。



Microsoft SQL Server 2000 Developer Edition

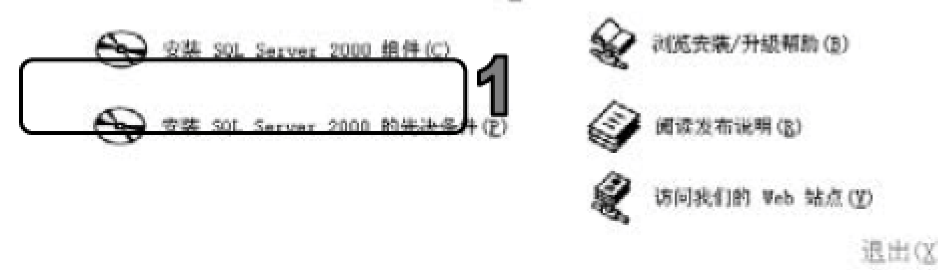


图 11-1 SQL Server 2000 安装向导



图 11-2 安装 SQL Server 2000 组件

(3) 将鼠标移至可选择组件上方，会出现相应的描述信息。单击【安装数据库服务器】进行数据库服务器的安装。此时，安装程序启动，在欢迎页面中单击【下一步】按钮，弹出【计算机名】对话框，如图 11-3 所示。

(4) 选择【本地计算机】单选按钮，单击【下一步】按钮，进行安装选择，如图 11-4 所示。



图 11-3 指定计算机名

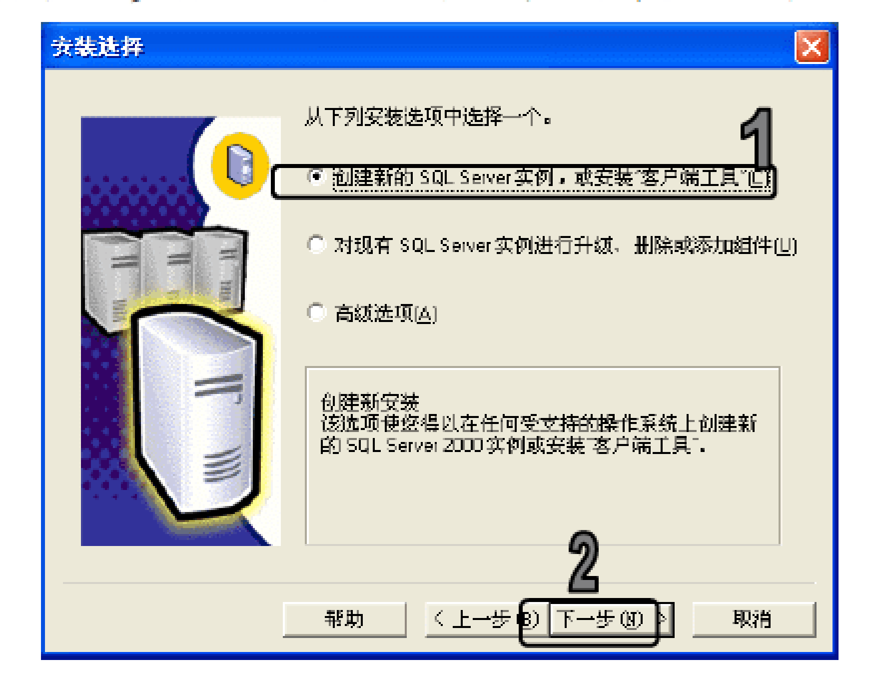


图 11-4 安装选择





(5) 选择【创建新的 SQL Server 实例，或安装客户端工具】单选按钮，单击【下一步】按钮输入用户信息，如图 11-5 所示。

(6) 输入【姓名】和【公司】(可不用输入)，单击【下一步】按钮，弹出【软件许可证协议】对话框，如图 11-6 所示。

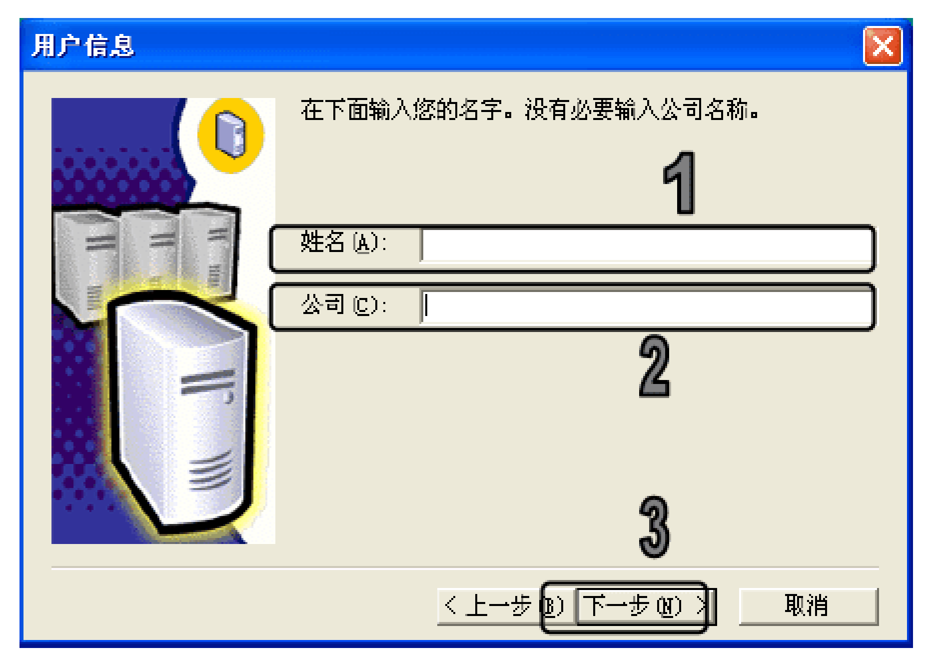


图 11-5　用户信息

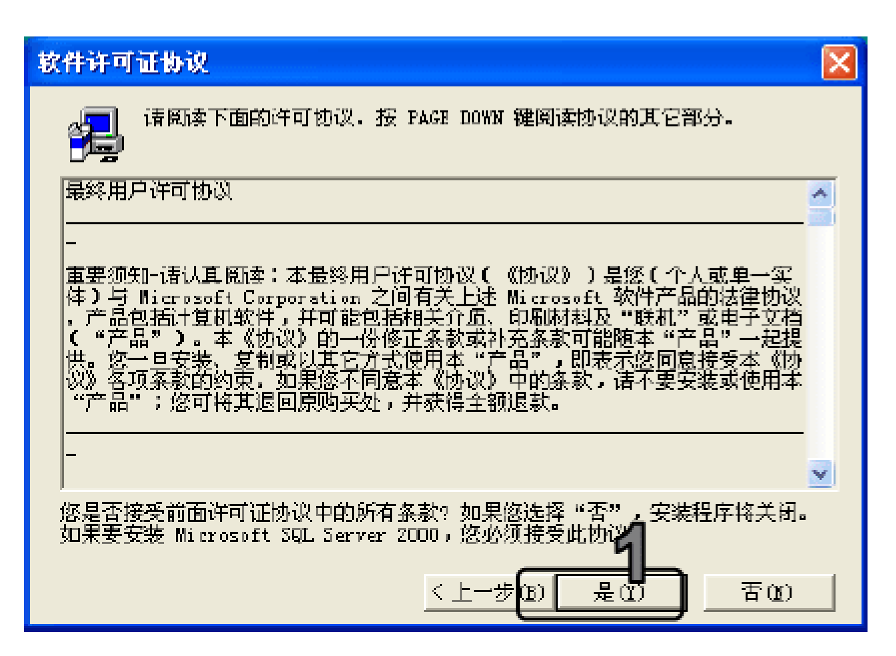


图 11-6　软件许可证协议



(7) 单击【是】按钮同意软件许可证协议，接着弹出【安装定义】对话框，如图 11-7 所示。

(8) 选择【服务器和客户端工具】单选按钮，单击【下一步】按钮定义实例名，如图 11-8 所示。

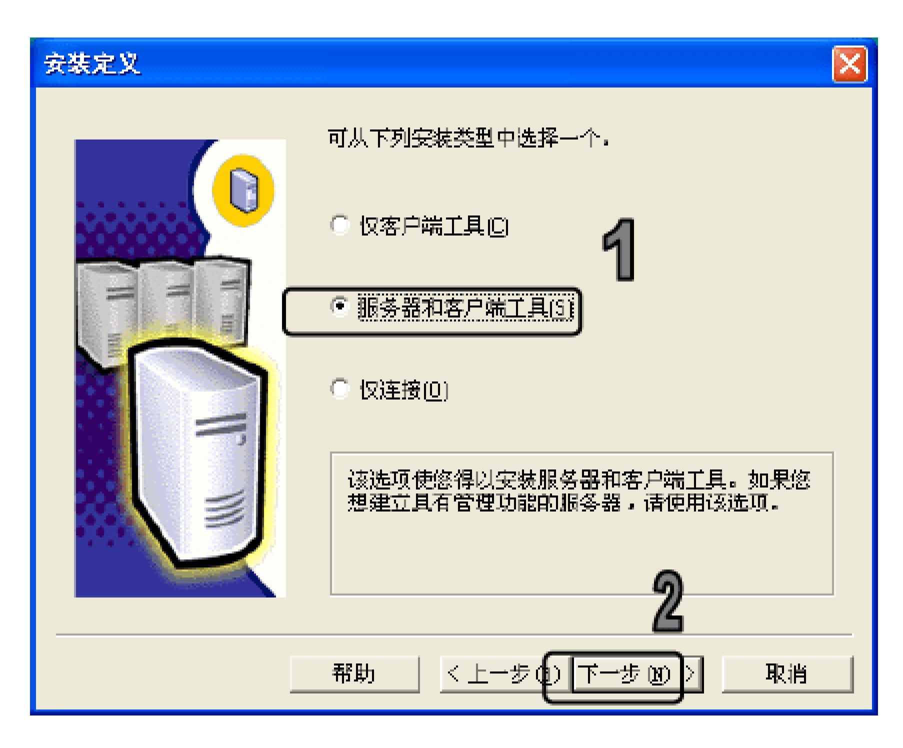


图 11-7　安装定义

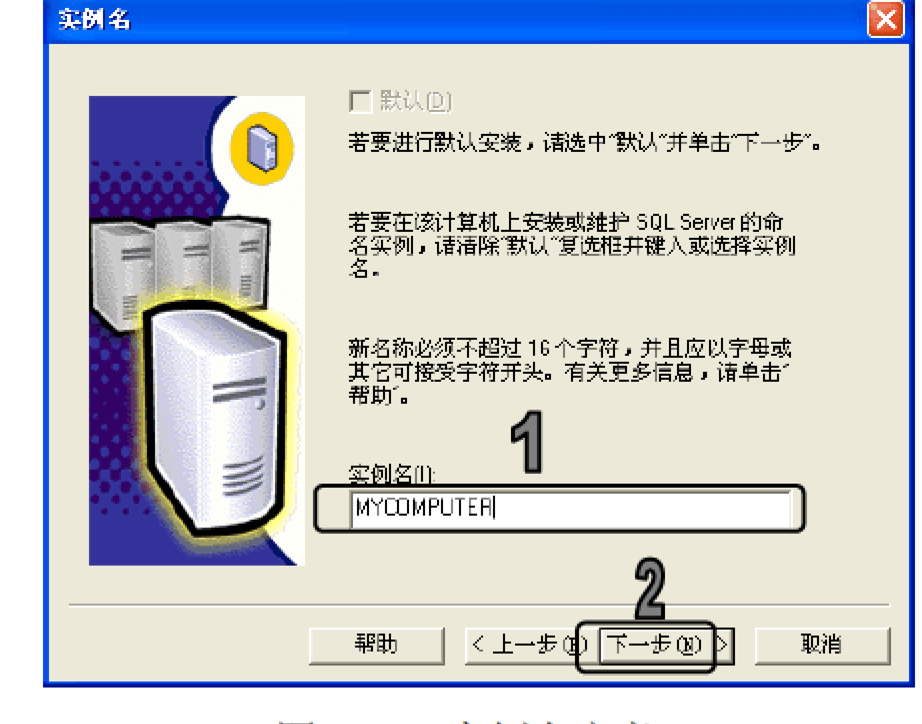
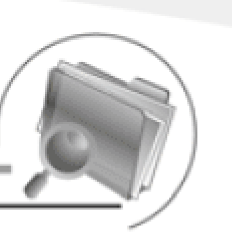


图 11-8　实例名定义

(9) 实例名事实上是数据库服务器的一个别名，用于系统识别相应的服务器，可以设置为对数据库应用有意义的值或任意值，如机器名等，单击【下一步】按钮选择安装类型，如图 11-9 所示。

(10) 选择【典型】单选按钮，通过单击【浏览】按钮分别选定【程序文件】和【数据文件】的安装位置。在该对话框的下方会显示安装需要的磁盘空间和相应驱动器上的可用空间以作提示。单击【下一步】按钮确定账户，如图 11-10 所示。



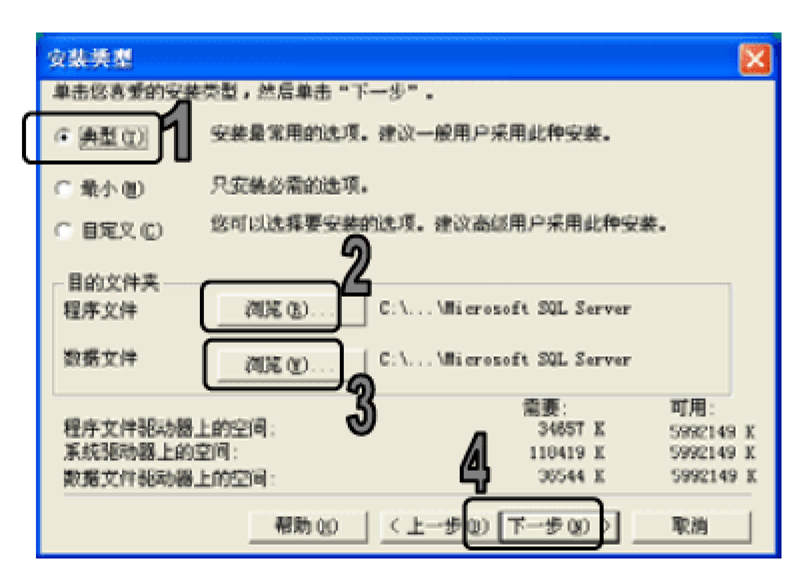


图 11-9　安装类型选择

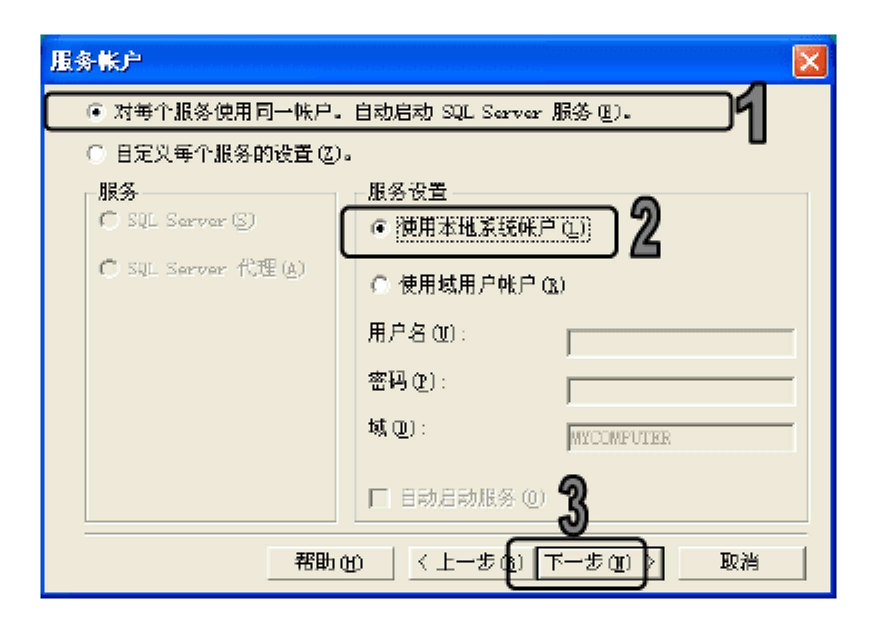


图 11-10　服务账户指定

(11) SQL Server 安装完之后，会按照 Windows 服务的方式自动启动 SQL Server 服务。选择【对每个服务使用同一账户，自动启动 SQL Server 服务。】单选按钮。在【服务设置】选项组中，选定【使用本地系统账户】单选按钮。单击【下一步】按钮定义身份验证模式，如图 11-11 所示。

(12) 身份验证模式是指 SQL Server 系统安全性验证的方式。SQL Server 2000 提供了两种身份验证模式：使用 Windows 账户的方式和采用 SQL Server 自定义账户的方式。这里选择【混合模式(Windows 身份验证和 SQL Server 身份验证）】，单击选中【空密码】复选框，将数据库系统管理员 sa 的密码设置为空。单击【下一步】按钮，弹出【开始复制文件】对话框，如图 11-12 所示。

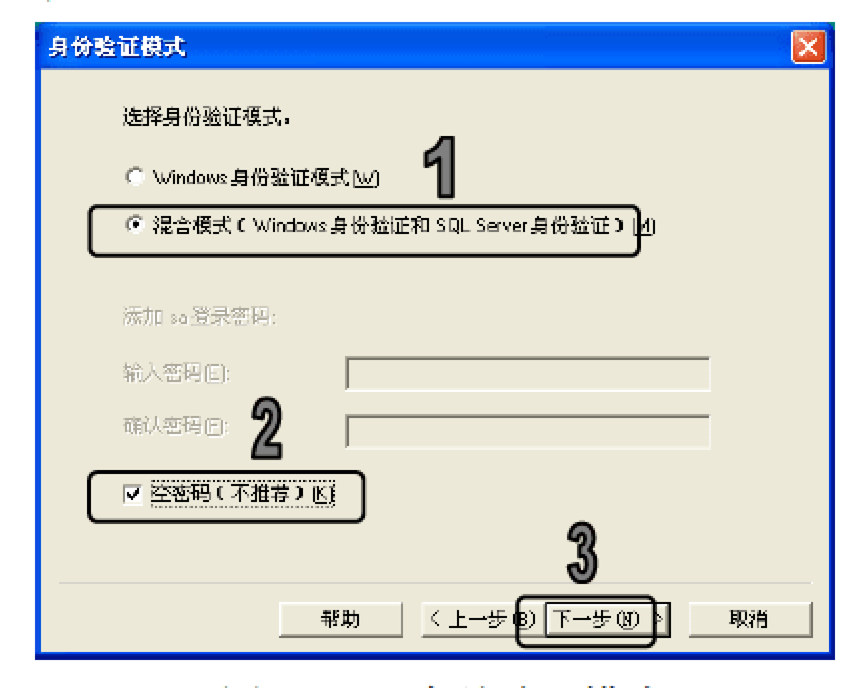


图 11-11　身份验证模式

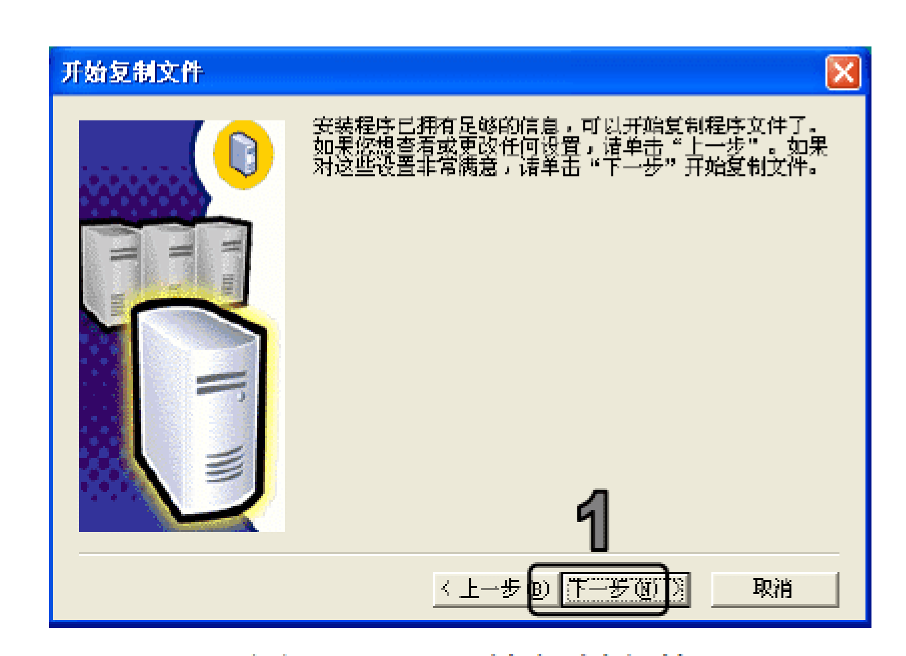


图 11-12　开始复制文件

(13) 至此，SQL Server 2000 安装过程中所需的配置都已设置完毕，单击【下一步】按钮开始复制文件进行安装的过程，后面的安装进程会自动进行直至完成。安装完成之后可以看到在【程序】菜单中增加了【Microsoft SQL Server】组，至此 SQL Server 2000 的安装已经完成。

11.5.2　SQL Server 2000 企业管理器

SQL Server 2000 中最常用的工具就是企业管理器。通过企业管理器可以方便快捷地创建数据库系统，使用户在对 SQL 不是很精通的情况下，一样可以建立一个良好的数据库。





企业管理器是一个集成化的数据操作环境，几乎所有的数据库操作都可以在这里完成。它是 SQL Server 2000 的主要管理工具，提供了一个遵从微软管理控制台(MMC)的用户界面，用户可以进行如下操作：

- 定义运行 SQL Server 的服务器组。
- 将个别服务器注册到组中。
- 为每个已注册的服务器配置所有的 SQL Server 选项。
- 在每个已注册的服务器中创建并管理所有 SQL Server 数据库、对象、登录、用户和权限。
- 在每个已注册的服务器上定义并执行所有 SQL Server 管理任务。
- 通过调用 SQL 查询分析器，交互地设计并测试 SQL 语句、批处理和脚本。
- 调用为 SQL Server 定义的各种向导。

单击 Windows 的【开始】|【程序】|【Microsoft SQL Server】|【企业管理器】命令，即可打开企业管理器。在窗口左侧的树状目录栏中，展开 Microsoft SQL Servers 节点，界面如图 11-13 所示。



通过企业管理器，可以进行创建数据库、执行数据库备份、执行各种向导、服务器配置、数据复制等各种操作，可以说掌握了企业管理器，就基本上掌握了 SQL Server 的所有操作。企业管理器窗口主要分成两部分，即菜单和工具条部分。【窗口】菜单用来管理窗口的布局以及窗口的数量。【操作】和【工具】菜单提供了当前可以进行的数据操作以及各种操作的工具，工具栏中的各种图标都直观地表现出了其功能，读者可以很容易使用。

树状区主要包括了 SQL Server 系统的各种对象和服务，如图 11-14 所示，包括数据库的管理、表和视图的管理、安全性事务、数据转化等。

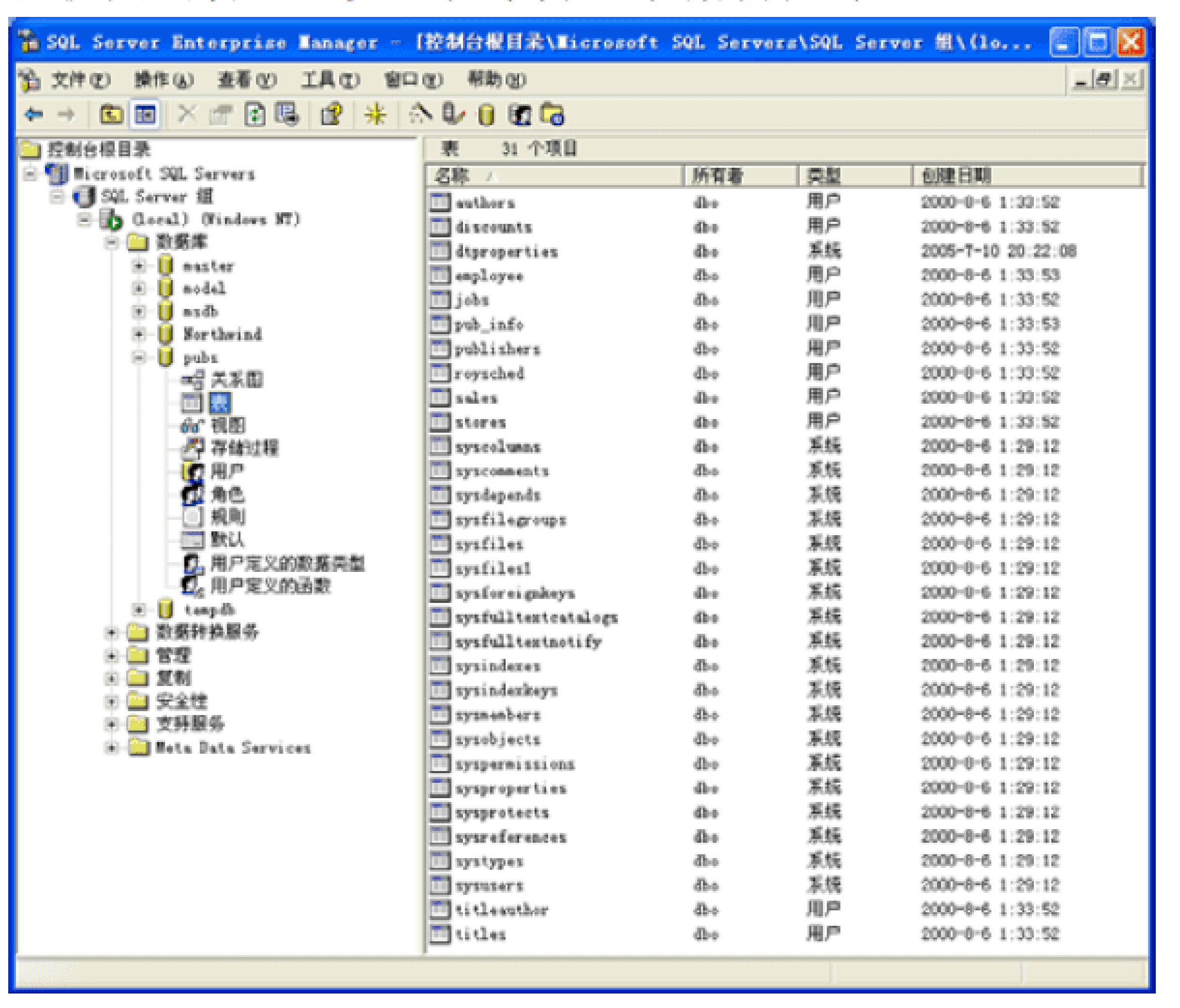


图 11-13 企业管理器

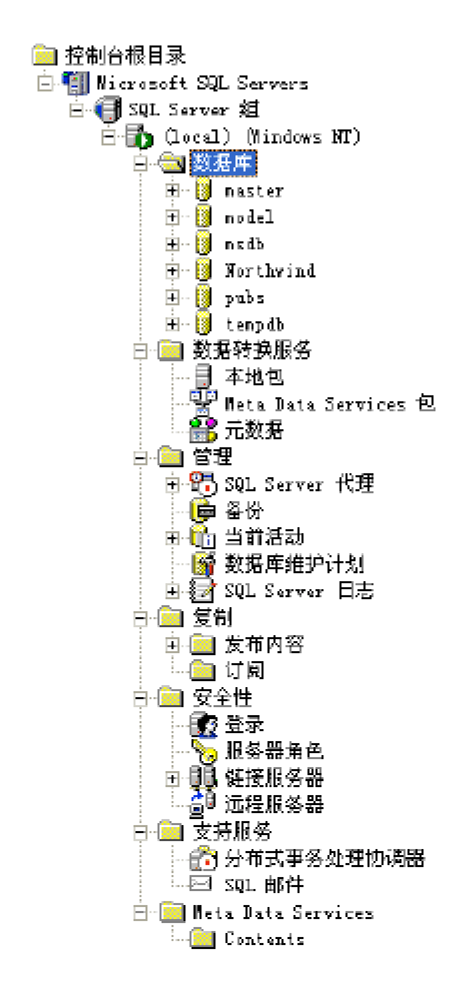


图 11-14 树状区

右边任务区是进行图形化操作的地方，其内容随着当前对象的不同而不同，有数据库的管

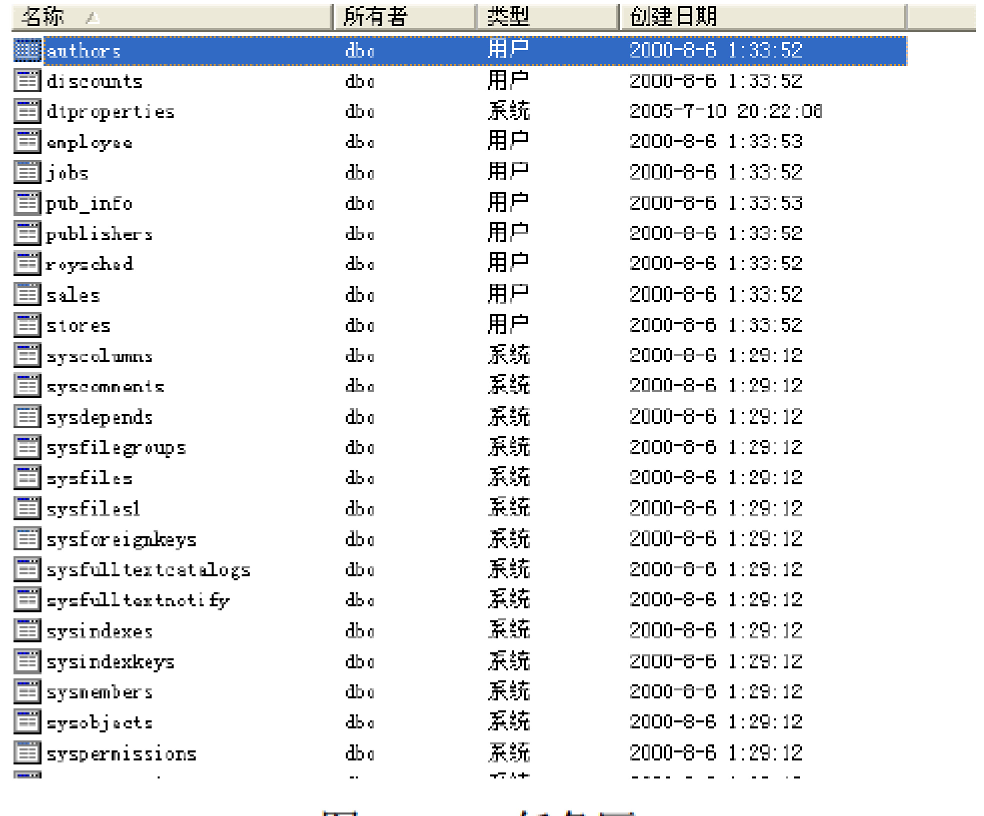


理、表和视图的操作、数据转换、安全性等图形化操作方式，比如当在树状区中选择的对象是某一数据库时，该区为数据库的表的基本信息，如图 11-15 所示。熟悉了界面之后，就要用它来进行数据库操作。

下面简单介绍如何利用 SQL Server 2000 的企业管理器创建数据库和表，如何操作数据以及如何使用 SQL 语句进行相应的操作。

⊙ 创建数据库

(1) 在树状区中右键单击【数据库】，从弹出的快捷菜单中选择【新建数据库】命令，如图 11-16 所示。



名称	所有者	类型	创建日期
authors	dbo	用户	2000-8-6 1:33:52
discounts	dbo	用户	2000-8-6 1:33:52
dtproperties	dbo	系统	2005-7-10 20:22:08
employee	dbo	用户	2000-8-6 1:33:53
jobs	dbo	用户	2000-8-6 1:33:52
pub_info	dbo	用户	2000-8-6 1:33:53
publishers	dbo	用户	2000-8-6 1:33:52
roysched	dbo	用户	2000-8-6 1:33:52
sales	dbo	用户	2000-8-6 1:33:52
stores	dbo	用户	2000-8-6 1:33:52
syscolumns	dbo	系统	2000-8-6 1:29:12
syscomments	dbo	系统	2000-8-6 1:29:12
sysdepends	dbo	系统	2000-8-6 1:29:12
sysfilegroups	dbo	系统	2000-8-6 1:29:12
sysfiles	dbo	系统	2000-8-6 1:29:12
sysfiles1	dbo	系统	2000-8-6 1:29:12
sysforeignkeys	dbo	系统	2000-8-6 1:29:12
sysfulltextcatalogs	dbo	系统	2000-8-6 1:29:12
sysfulltextnotify	dbo	系统	2000-8-6 1:29:12
sysindexes	dbo	系统	2000-8-6 1:29:12
sysindexkeys	dbo	系统	2000-8-6 1:29:12
sysmembers	dbo	系统	2000-8-6 1:29:12
sysobjects	dbo	系统	2000-8-6 1:29:12
syspermissions	dbo	系统	2000-8-6 1:29:12

图 11-15　任务区



图 11-16　创建数据库

(2) 在弹出的如图 11-17 所示的【数据库属性】对话框中，输入数据库【名称】(如 Test)，单击【确定】按钮即可。

⊙ 创建表

(1) 在树状区中展开刚刚创建的数据库 Test，右键单击【表】，从弹出的快捷菜单中选择【新建表】命令，如图 11-18 所示。

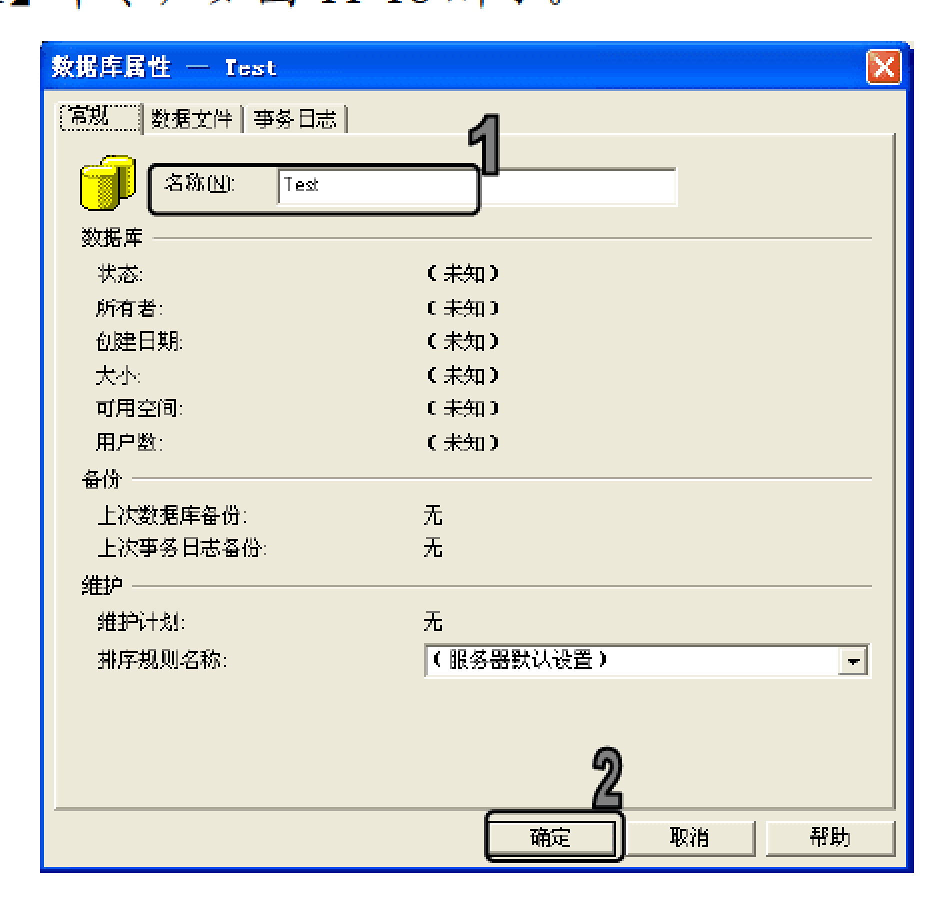


图 11-17　数据库属性

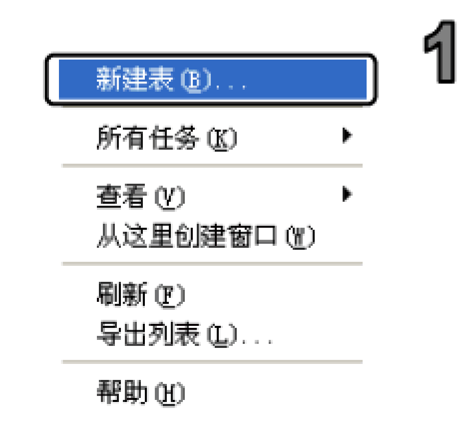
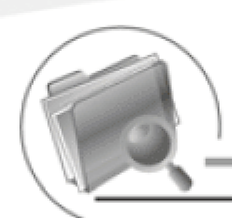


图 11-18　创建表





(2) 在打开的窗口中，输入表的列名，选择数据类型并指定数据长度，如图 11-19 所示。

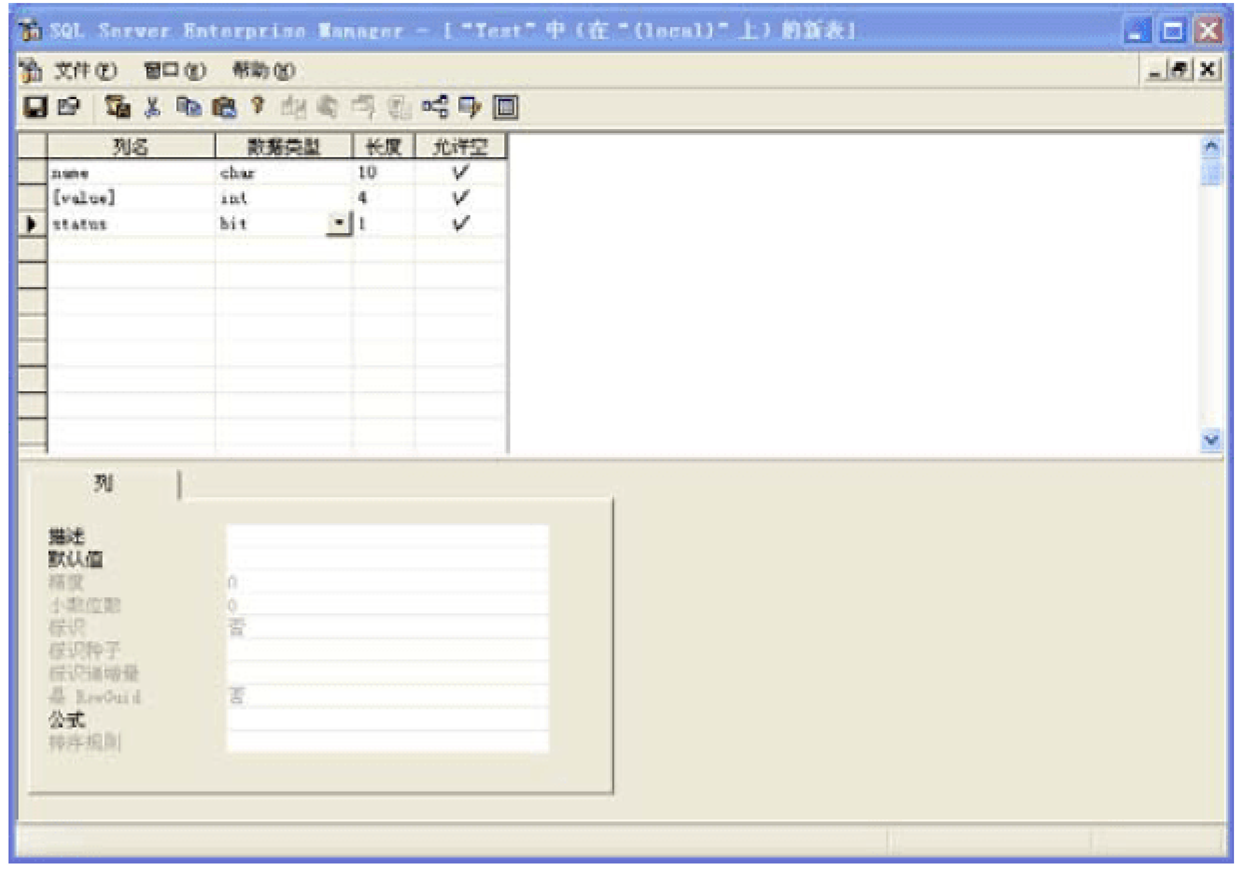


图 11-19　建立新表



(3) 所有列都创建完毕后，单击窗口工具栏中的保存按钮，将弹出如图 11-20 所示的【选择名称】对话框，输入表名后单击【确定】按钮即完成表的创建。

◉ 操作数据

(1) 在树状区中展开需要操作的表所在的数据库，选中【表】，在右侧的任务区会显示所有该数据库中的表。选中要操作的表，单击鼠标右键，在弹出的快捷菜单中选择【打开表】|【返回所有行】命令，如图 11-21 所示。



图 11-20　选择表名



图 11-21　打开表

(2) 如图 11-22 所示窗口将被打开，所有的数据都会被查询并显示出来，在这个窗口中可以非常方便地进行数据的增加、删除和修改等操作。

(3) 单击如图 11-22 所示窗口工具栏中的 SQL、网格或关系图窗格按钮(或在如图 11-21 所示的菜单中选择【打开表】|【查询】命令)，就可以非常方便地通过书写 SQL 或者更简单地图形化表和列来自定义查询。查询定义完后，单击工具栏中的【运行】按钮即可执行相应的 SQL 语句，如图 11-23 所示。

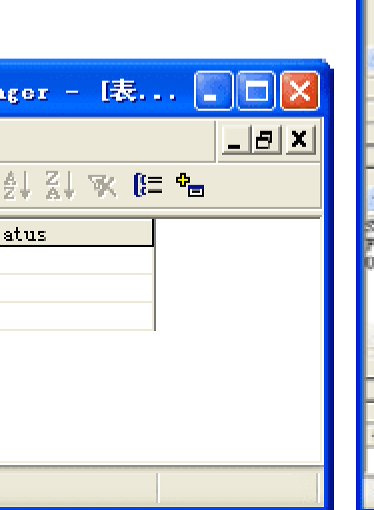


图 11-22　操作数据　　　　图 11-23　执行查询语句

11.5.3　SQL Server 2000 查询分析器

SQL Server 查询分析器是 SQL Server 用户操作界面的另一个主要组成部分，通过它同样可以对数据库进行操作。选择 Windows 任务栏中的【开始】|【程序】|【Microsoft SQL Server】|【查询分析器】命令，打开【查询分析器】窗口。

查询分析器是 SQL Server 2000 中分析并执行 SQL 语句的软件工具，用它可以执行输入的 SQL 语句或者脚本文件。启动查询分析器时会出现连接对话框，要求输入 SQL Server 服务器和账户信息，如图 11-24 所示。



图 11-24　连接主机对话框

输入正确信息后，单击【确定】按钮即可进入查询分析器界面。如图 11-25 所示，查询分析器除菜单栏之外由 4 个部分组成，即对象浏览器、脚本编辑窗口、工具条和执行结果显示窗口。在中间部位的下拉框中选择数据库为 Test，在查询脚本编辑窗口中输入查询 SQL 语句，单击【执行查询按钮】即可在下方的窗口中看到查询结果。



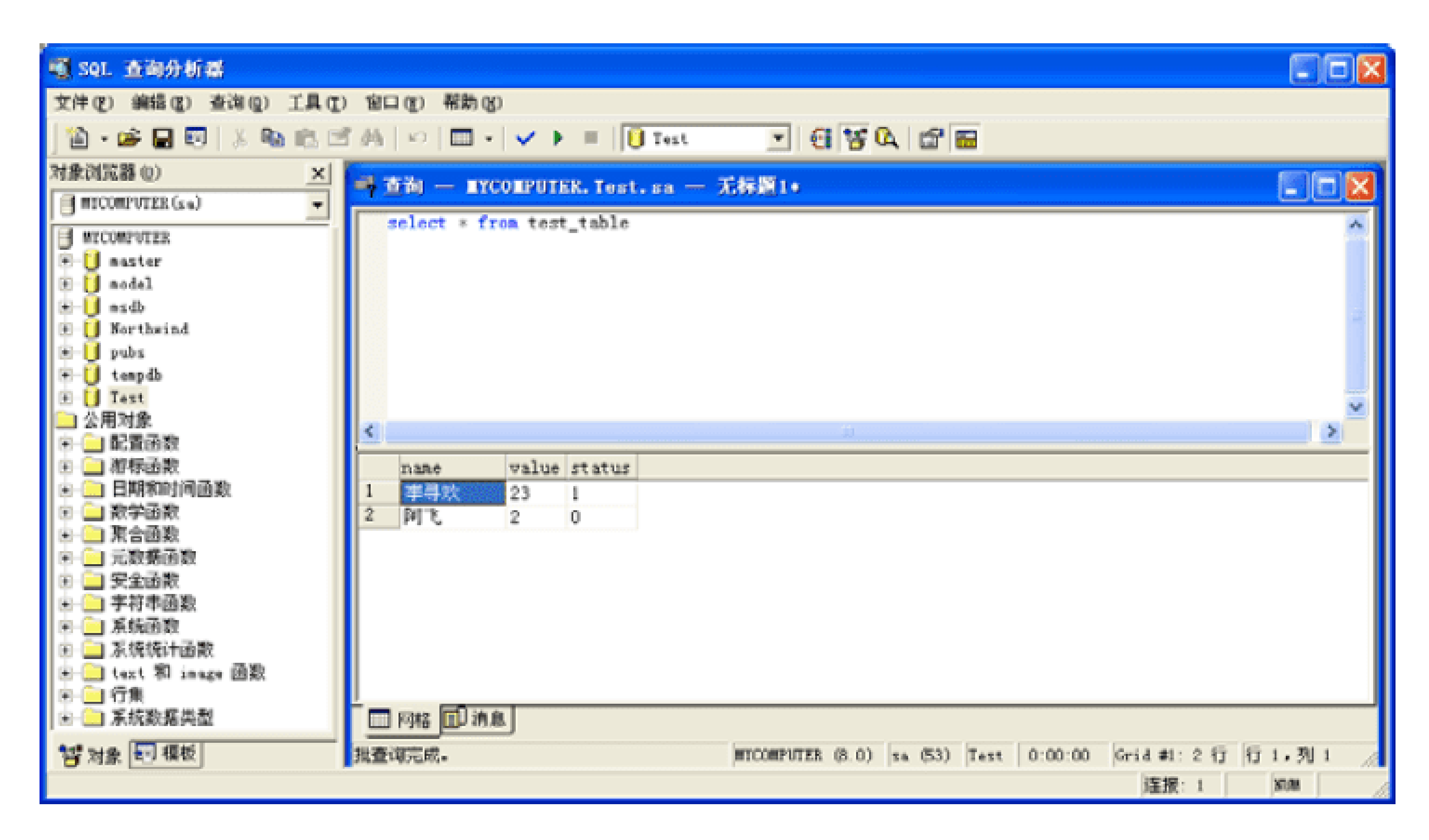


图 11-25　查询分析器



11.6　上机练习

本章上机实验主要以学习 SQL 语句和练习使用 SQL Server 2000 的企业管理器和查询分析器为主。其中，读者应重点掌握如何安装 SQL Server 2000，Select、Insert、Update 和 Delete 等 SQL 语句的使用，使用企业管理器创建数据库、创建表和对数据进行操作，使用查询分析器分析和执行 SQL 语句。

下面以前面介绍的 company 数据库为例进行上机练习。

(1) 选择【开始】|【所有程序】|【Microsoft SQL Server】|【企业管理器】命令启动 SQL Server 2000 企业管理器。

(2) 展开树状图节点【SQL Server 组】，连接数据库服务器。

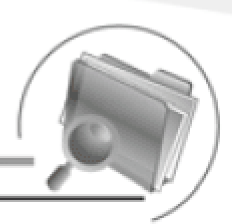
(3) 右键单击【数据库】，在弹出的快捷菜单中选择【新建数据库】命令。

(4) 在弹出的【数据库属性】对话框中输入数据库名 company。单击【确定】按钮完成数据库的创建。

(5) 展开 company 数据库项目，右键单击【表】，在弹出的快捷菜单中选择【新建表】命令。

(6) 在打开的新表中，输入以下 SQL 语句对应的行信息。

```
create table employee(
  empno varchar(8) primary key,
  empname    varchar(10),
  deptno     varchar(2),
  m_salary int,
  job          varchar(20),
  commission    int
)
```



(7) 单击存盘按钮，输入表名 employee 并退出。

(8) 再以同样方式创建 department 表。

(9) 在任务区右击 employee 表或 department 表，选择【打开表】|【返回所有行】命令。

(10) 在打开的数据窗口中，录入一批数据。

(11) 退出企业管理器。

(12) 选择【开始】|【所有程序】|【Microsoft SQL Server】|【查询分析器】命令，在【连接到 SQL Server】对话框中，选择本地实例进行连接。

(13) 在【查询分析器】中选择 company 数据库。

(14) 若在前面没有录入测试数据，也可以在【查询分析器】中执行如下 insert 语句代替。

```
insert into department (deptno,location) values ('10','北京');
insert into department (deptno,location) values ('20','上海');
insert into department (deptno,location) values ('30','成都');
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('1','老王','10',3000,'salesman',2800) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('2','张三','20',3300,'salesman',3000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('3','李四','10',5200,'clerk',2500) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('4','王二麻子','10',3500,'salesman',1500) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('5','老陈','10',6300,'manager',5000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('6','老郑','20',6500,'manager',6000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('7','小潘','20',5500,'clerk',3500) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('8','小黄','30',2500,'clerk',2000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('9','小余','30',1500,'salesman',2000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('10','老沈','30',4500,'manager',5000) ;
insert into employee (empno,empname,deptno,m_salary,job,commission)
values ('11','小金','10',1200,'salesman',null) ;
```

(15) 书写 Select、Update 和 Delete 语句。

(16) 选择【查询】|【分析】命令对 SQL 语句进行分析，调试错误信息修改语句。

(17) 选择【查询】|【执行查询】命令执行 SQL 语句。



11.7 习题

11.7.1 填空题

1. 实体之间的联系有：____________、____________和____________。

2. SQL 语言由____________、____________和____________3 种元素构成。

3. SQL Server 2000 有 2 种身份验证模式，分别是____________和____________。

4. 数据操作语言包括________、________、________和________4 种语句。



11.7.2 选择题

1. 以下 SQL 语句中，(　　　)属于数据定义语言。

A. Grant　　B. Select　　C. Create　　D. Delete

2. 以下 SQL 语句中，(　　　)属于数据操纵语言。

A. Grant　　B. Select　　C. Create　　D. Drop

3. 以下 SQL 语句中，(　　　)属于数据控制语言。

A. Grant　　B. Select　　C. Create　　D. Delete

4. 以下 SQL 语句中，语法正确的是(　　　)。

A. delete * from employee

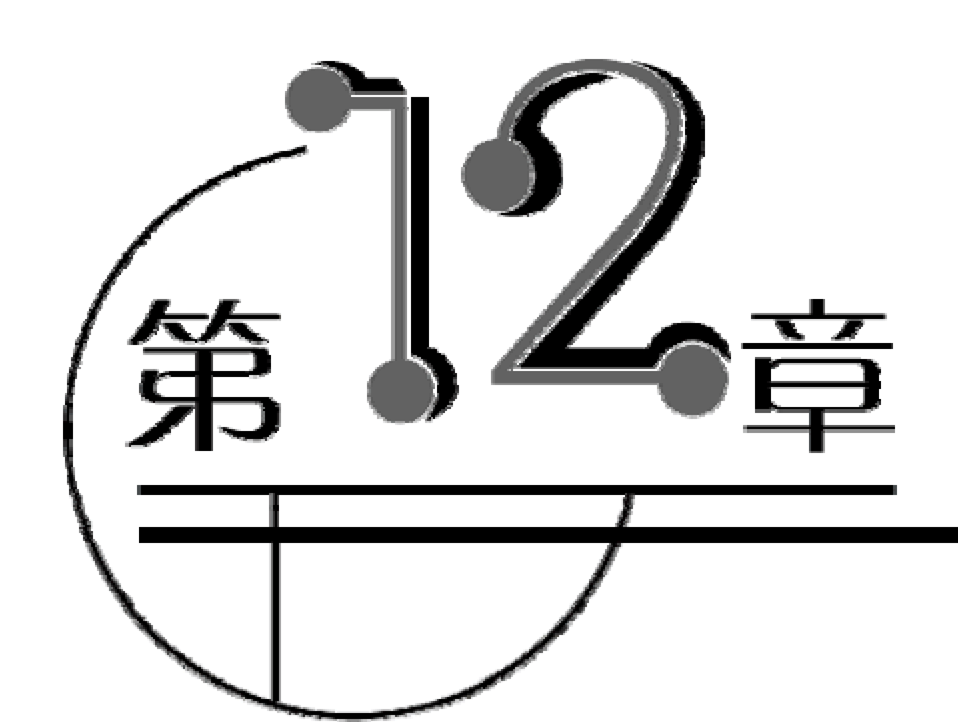
B. select * from employee where m_salary>5000

C. insert ('10'，' 香港 ') into department

D. update from employee set commission=3000

11.7.3 问答题

1. 关系模型有什么优点？
2. 为什么要使用存储过程？它有什么优点？
3. 简述视图的概念和作用。



第12章 JSP 数据库应用

学习目标

目前，几乎很少有网站采用纯粹的静态页面，很难想象当今的任何商业应用可以不使用数据库。对于使用 JSP 技术开发网络应用的程序员来说，数据库应用的学习是其必修课程。

JDBC(Java 数据库连接)是 Java 提供的一种开放的数据库连接解决方案。利用 JDBC 的 API，Java 应用程序(包括 JSP 应用)就可以使用同样的语法轻松访问不同的关系型数据库，使程序员只需考虑业务逻辑，而不用花大量的时间和精力去研究如何从不同的数据库系统中读取和存放数据。本章将深入介绍 JDBC 的相关知识以及如何在 JSP 中使用 JDBC 来访问数据库。

本章重点

- 数据库驱动程序
- JDBC 核心 API
- 使用 JDBC 访问数据库
- 数据库事务

12.1 JDBC 简介

JDBC 技术的鼻祖是 Microsoft 公司提供的数据库驱动程序 API：ODBC(开放式数据库连接)。Microsoft 为了提供在 Windows 平台上方便地访问各种数据库资源，建立了标准的数据库访问 API，它是厂商驱动程序、平台和数据库之间的中介。使用 ODBC，并不需要在客户程序中嵌入 SQL，而是定义了一套用于直接访问数据库的函数。JDBC 的 API 就是基于 ODBC 开发而来的。

通过 JDBC 可以将 SQL 语句传送给任何数据库，并返回相应的数据结果。JDBC 的体系结构如图 12-1 所示。

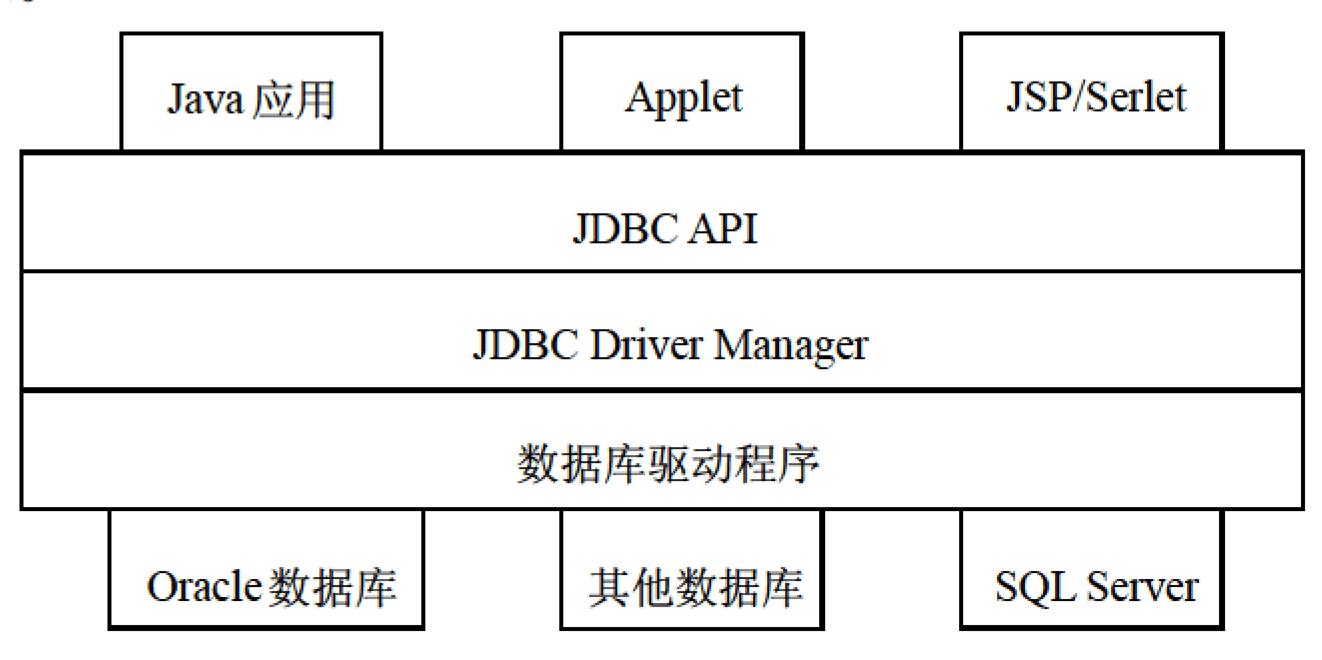


图 12-1　JDBC 体系结构

下面分别介绍数据库驱动程序和 JDBC API，JDBC Driver Manager 将在后面使用 JDBC 访问数据库时再详细介绍。

12.1.1　数据库驱动程序

一般情况下，数据库系统供应商都会提供用于访问数据库服务器所管理数据的 API，JDBC API 则提供了统一的用户访问界面。虽然简化了用户的学习成本，提高了开发效率，但是在 JDBC 的底层实现中，必须将 JDBC 的 API 与厂商的数据库访问 API 结合起来才有可能获取数据库信息。这就是数据库驱动程序产生的原因，JDBC 驱动程序成为 JDBC API 和相关厂商 API 联系的桥梁。下面是 JDBC 数据库驱动程序的几种类型。

- 类型 1：JDBC-ODBC 桥

JDBC-ODBC 桥将 JDBC 调用转换成相应的 ODBC 调用，通过 ODBC 库访问 ODBC 数据源。这种方式由于每次进行数据库访问调用时，都要经过多个层次的调用，因此显得效率较低。同时，应用程序需要安装和配置 JDBC-ODBC 桥 API、ODBC 驱动程序和本地数据库 API 才能执行。不过，JDBC-ODBC 桥提供了一种方便实用的方式，因为 ODBC 已经发展成为一种标准，在多数操作系统(尤其是 Windows 系统)中都预先安装和配置了常用数据库系统的驱动。在访问某些数据源，如 Microsoft Access 数据库时，这是唯一可行的方式。

- 类型 2：部分 Java 驱动，部分本地驱动

这种方式需要在客户机上安装本地 JDBC 驱动程序和具体厂商的本地数据库 API。本地 JDBC 驱动程序部分使用 Java 代码处理访问请求，部分使用本地化代码访问厂商的数据库 API 以转化数据库调用。这种方式提高了效率，而且能够充分利用厂商 API 提供的所有功能。

- 类型 3：中介数据库服务器

这是一种非常灵活的方式，在纯 Java 的 JDBC 驱动程序和数据源之间建立中介数据库服务器(中间件)。中介服务器作为一个或多个数据库服务器的网关，通过它可以连接不同的数据库



服务器。这样客户端开发人员就不用过多的考虑数据库连接细节，尤其是在开发复杂的应用程序(可能要访问多种不同数据库服务器)时可以很方便地实现。

◉ 类型 4：本地协议纯 Java 驱动

这种驱动程序通过与数据库建立直接的网络套接字连接，采用具体厂商的本地网络协议将 JDBC 调用转换为直接的网络调用。这种方式效率最高，而且部署采用这种驱动程序方式的应用程序也更方便，它不需要安装其他的运行库和中间件。目前，多数的数据库厂商都提供自己的驱动程序。

在本书中，为简单起见，将主要采用类型 1 的方式访问 Microsoft SQL Server 数据库。使用其他方式都需要去相应的供应商处下载合适的 JDBC 驱动程序并安装。

12.1.2　JDBC 核心 API



目前，JDBC API 已经发展到 3.0 版，它包括 JDBC 核心 API(JDBC Core API)和 JDBC 可选包 API(JDBC Optional Package API)。在 JDK1.5 中，核心 API 是指 java.sql 包中的所有类和接口，而可选包 API 则包括 javax.sql，javax.rowset，javax.rowset.serial 和 javax.rowset.spi 几个包中的内容。

表 12-1 显示了 java.sql 包中所有的类和接口。

表 12-1　JDBC 核心 API

类/接口	含　义
Array	接口，数组，用于表示 SQL 数据类型和 Java 类型之间的映射
Blob	接口，大二进制对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Clob	接口，大字符对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Date	类，日期对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Ref	接口，引用对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Struct	接口，对象数组，用于表示 SQL 数据类型和 Java 类型之间的映射
Time	类，时间对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Timestamp	类，时间戳对象，用于表示 SQL 数据类型和 Java 类型之间的映射
Types	类，定义了各种 SQL 数据类型
Connection	接口，创建和管理数据库连接
Driver	接口，所有数据库驱动程序都会实现 Driver 接口
DriverInfo	默认访问类(只可由同一包或派生类访问)，获取 Driver 信息
DriverManager	类，对数据库驱动程序进行管理
DriverPropertyInfo	类，获取数据库驱动程序属性信息
DatabaseMetaData	接口，获取数据库元数据
ParameterMetaData	接口，获取参数值元数据
ResultSetMetaData	接口，获取结果集元数据

(续表)

类/接口	含　义
ResultSet	接口，结果集
Savepoint	接口，表示事务中的一个存储点
SQLData	接口，表示用户定义的 SQL 类型映射到 Java 类
SQLInput	接口，用户定义的 SQL 输入对象
SQLOutput	接口，用户定义的 SQL 输出对象
SQLException	类，SQL 异常
BatchUpdateException	类，批处理更新异常，SQLException 子类
SQLWarning	类，SQL 警告，SQLException 子类
DataTrunction	类，SQLWarning 子类，当 JDBC 意外地截取一个数据值时抛出的异常
SQLPermission	final 类(不可派生)，提供安全访问
Statement	接口，管理 SQL 语句的建立和执行
PreparedStatement	接口，实现 Statement 接口，它是一个空白的 SQL 模板，可以设置未知参数
CallableStatement	接口，实现了 PreparedStatement 接口，用于调用存储过程



表 12-1 中的所有类都在 JDK 所带的基础包中被实现，而相关的接口也都有厂商的数据库驱动程序来实现，因此使用上述的这些类和接口的方法以及对应的驱动就可以方便地存取各种数据库数据。在后面的章节中，将讨论这些 API 中比较常用的类和接口，通过实例来介绍如何加载数据库驱动、打开数据库连接、执行 SQL、获取结果集以及处理 SQL 异常。

12.1.3　JDBC 可选包 API

如 12.1.2 节所述，JDBC 可选包 API 包括 javax.sql、javax.rowset、javax.rowset.serial 和 javax.rowset.spi 这 4 个包中的内容。它主要包括以下功能：

⊙　连接池

在 12.1.2 节中提到连接(Connection)，使用核心包 API，在每次进行数据库访问时需要在内存中实例化一个新的连接对象，同时调用底层的数据库驱动与数据库服务器之间建立连接，而在使用完之后需要断开连接，销毁连接对象。对于 JSP 应用程序来说，这是一个非常耗费资源的操作。因此，在可选包中引入了连接池的概念，连接池就是数据库连接的缓存，连接对象可以被重复使用，而不需要重复地创建和销毁。因此使用连接池可以提高应用的整体性能。

⊙　基于 JNDI(Java 命名和目录接口)的数据库查询

采用基于 JNDI 的查询，用逻辑名来访问数据库资源，这样就不用每个客户都试图在本地虚拟机上装载数据库驱动程序。而且方便数据库访问资源的集中管理，避免了底层数据库的改变影响到客户端。



◉ 分布式事务

在某些应用中，需要采用多个语句来执行一个功能，而这些语句必须要么都成功，要么都失败，这就是事务的概念。最典型的事务的例子就是银行的转账应用，从一个账户转出的资金必须转入另一个账户，中间任何一步出现问题而导致资金不能入账都必须同时取消前一账户的转出。事务保证了数据的完整性和一致性。一般的数据库系统都能保证在同一连接中事务的完整性。

但是，在某些情况下，一个应用可能在单个的商业事务中采用不同的连接来访问相同的数据库，或者在单个事务中需要更新多个数据库中的数据。这就需要分布式事务的支持来保证数据的完整性和一致性。JDBC 可选包 API 提供了对分布式事务的强大支持。需要注意的是：分布式事务需要底层数据库系统的支持，并不是所有的数据库系统都能够支持。

◉ 行集(RowSet)

行集提供了与结果集(ResultSet)类似但功能更为强大的数据访问方法。行集可以支持离线模式，它并不需要在操作过程中保持一个连接，而是在对结果进行更新的情况下会采取适当的处理措施影响底层数据库。

由于这个包更多的被用于厂商实现他们的数据库访问接口，所以关于这部分的讨论主要集中在如何理解这些服务和概念，而不是如何进行应用开发。



12.2 使用 JDBC

本节将通过实例来介绍如何使用 JDBC 访问数据库，在这里主要学习 JDBC-ODBC 桥的方式。

12.2.1 配置 ODBC

为了使用 JDBC-ODBC 桥进行数据库连接，首先需要对 ODBC 进行一些附加设置，以便建立并使用它。下面简单介绍一下设置的过程。

(1) 单击【开始】|【设置】|【控制面板】命令，在打开的【控制面板】窗口中双击【管理工具】图标，打开【管理工具】窗口。双击【数据源(ODBC) 】图标打开【ODBC 数据源管理器】对话框，如图 12-2 所示。

(2) 选择【系统 DSN】选项卡，单击【添加(D)...】按钮，弹出如图 12-3 所示的【创建新数据源】对话框。

(3) 在驱动程序列表中选择【SQL Server】，单击【完成】按钮，弹出【创建到 SQL Server 的新数据源】对话框，如图 12-4 所示。

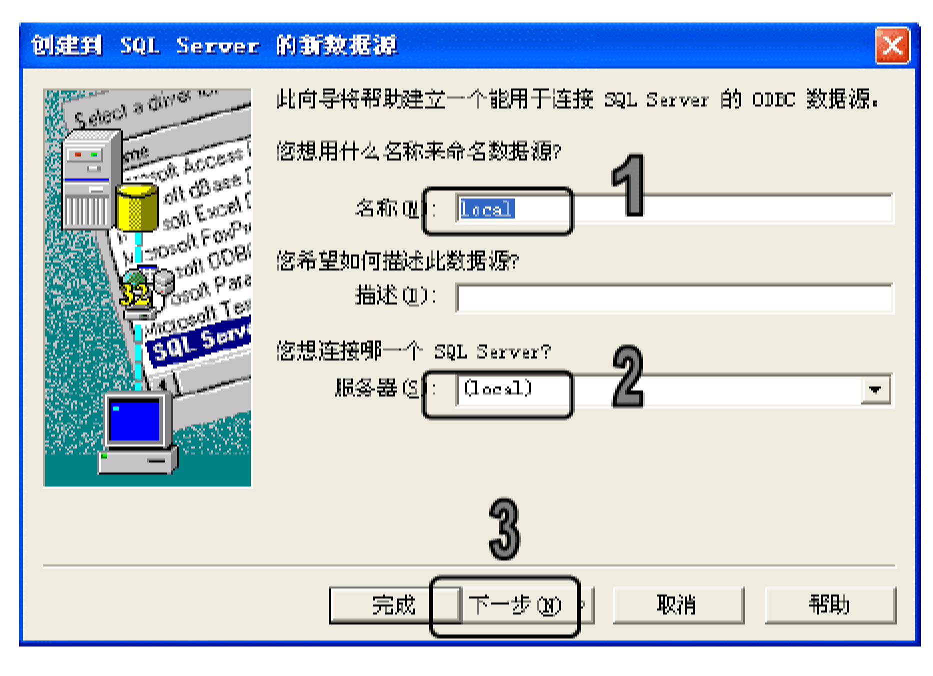
(4) 在【名称】文本框中输入数据源的名称，如 local，在【服务器】下拉列表框中选择服务器，如(local)。单击【下一步】按钮，将弹出用户验证对话框，如图 12-5 所示。







图 12-2　ODBC 数据源管理器　　　　图 12-3　【创建新数据源】对话框



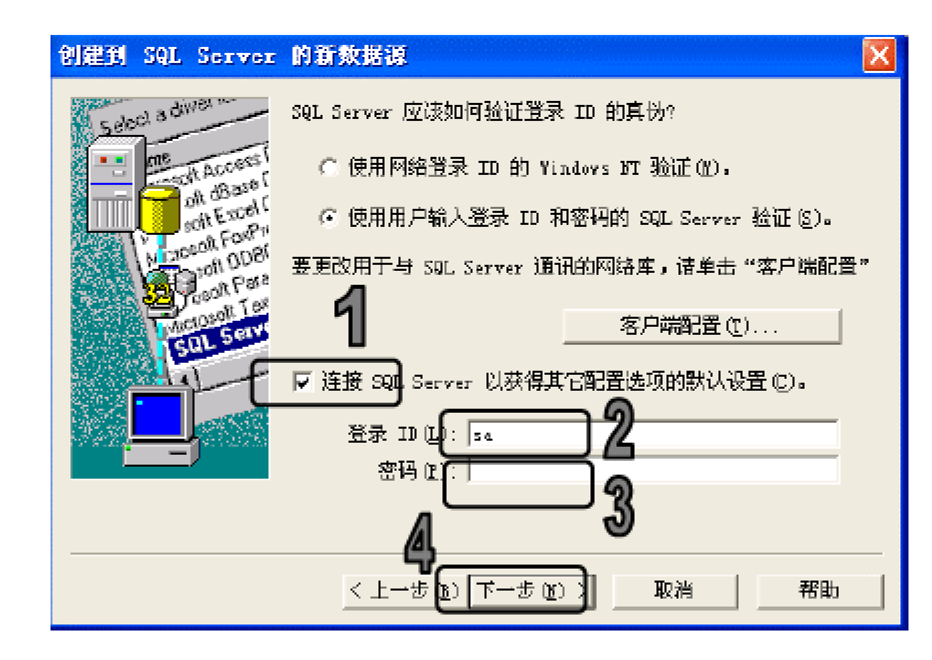


图 12-4　命名数据源　　　　图 12-5　用户验证

(5) 选中【连接 SQL Server 以获得其他配置选项的默认设置】复选框，在【登录 ID】文本框中输入登录名，如 sa，在【密码】文本框中输入密码。单击【下一步】按钮，对话框提示更改默认数据库等配置选项，如图 12-6 所示。

(6) 选中【更改默认的数据库为】复选框，在下面的下拉列表框中选择希望使用的数据库名，如本例选择的是 pubs 数据库。单击【下一步】按钮，在如图 12-7 所示的属性配置中保存原有配置即可。

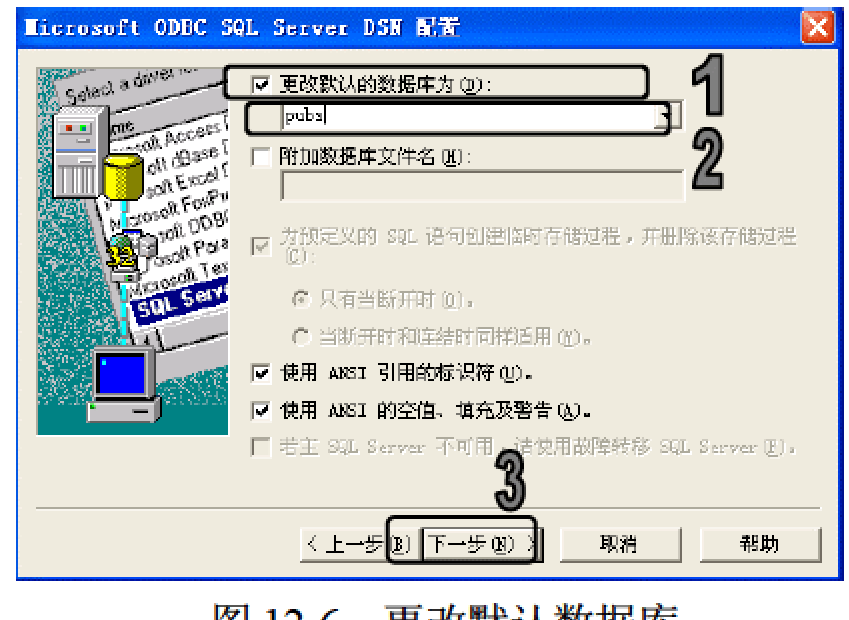




图 12-6　更改默认数据库　　　　图 12-7　更改属性

(7) 单击【完成】按钮，弹出如图 12-8 所示的【ODBC Microsoft SQL Server 安装】对话框。

(8) 单击安装对话框中的【确定】按钮即可完成所需的 ODBC 数据源的配置。不过在单击





【确定】按钮前可以先测试一下配置是否正确。单击【测试数据源】按钮，将会弹出如图 12-9 所示的【SQL Server ODBC 数据源测试】对话框，提示运行连接测试的结果，如果显示测试成功，单击【确定】按钮返回【ODBC Microsoft SQL Server 安装】对话框，单击【确定】按钮完成所有配置。否则，单击【取消】按钮回到如图 12-6 所示的配置对话框，重新进行正确的设置。

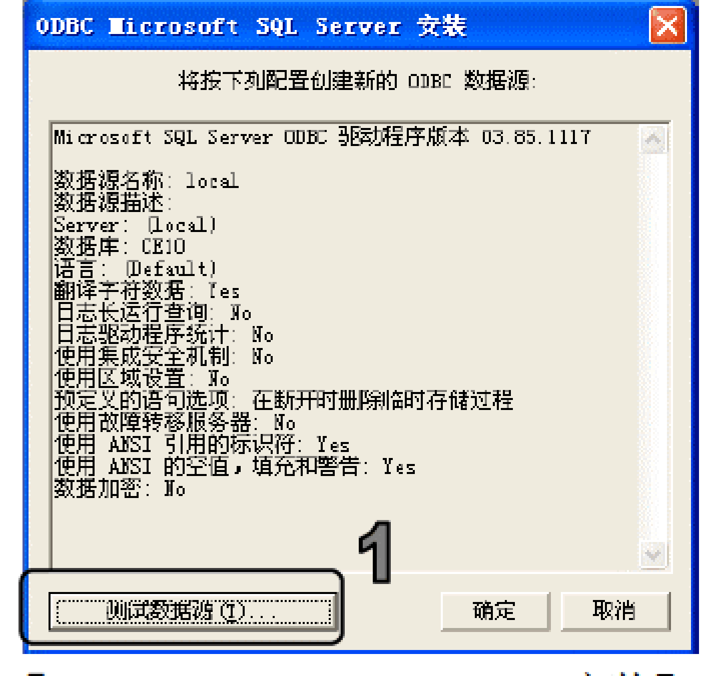


图 12-8　【ODBC Microsoft SQL Server 安装】对话框

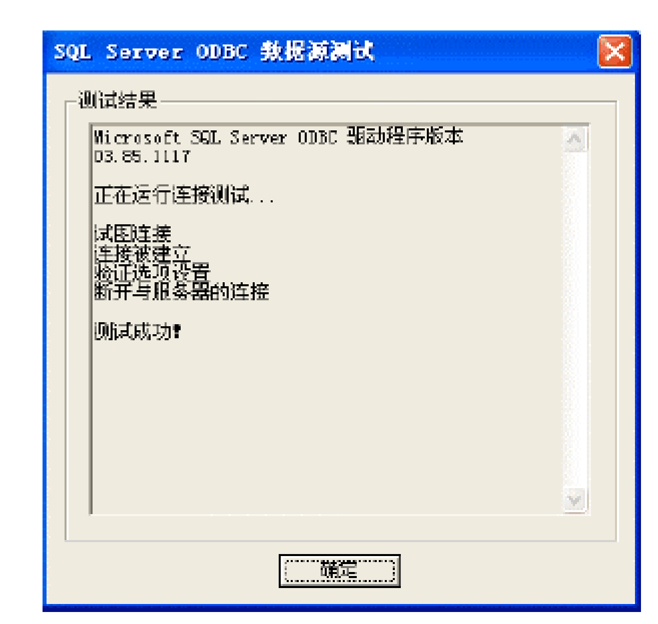


图 12-9　SQL Server ODBC 数据源测试对话框

完成上述的配置过程，就可以顺利地使用 JDBC-ODBC 桥来访问 SQL Server 数据库了。



12.2.2　使用 JDBC 访问数据库

使用 JDBC 访问数据库，主要包括以下 5 个步骤：

- 加载 JDBC 驱动程序
- 创建数据库连接
- 建立和执行数据处理语句
- 处理所得结果
- 关闭数据库连接

这个处理过程看起来比较复杂，但是 JDBC 却提供了非常简便的方法，上述的多数步骤都可以只用一两个 Java 语句就可以实现。下面先简要介绍各个步骤的实现，再用一个 JSP 实例进行说明。


提示

所有的 JDBC 方法调用都需要捕获 SQLException 异常，所以在 Java 代码中它们必须位于 try/catch 块中。不过如果直接在 JSP 中嵌入 Java 代码则可以不使用 try/catch，这是因为 JSP 在转换为 Servlet 之后会自动捕获和处理异常。

1. 加载 JDBC 驱动程序

前面介绍了 JDBC 必须要通过相应的 JDBC 驱动程序才能建立和数据库管理系统的联系，JDK 自带的库中包括了 JDBC-ODBC 桥驱动程序。访问数据库需要做的第一件事就是将合适的驱动程序加载到 Java 虚拟机上，使 Java 虚拟机能够通过驱动程序访问数据库。Java 虚拟机加

载类的方法是使用静态方法 Class.forName，如下面的代码表示加载 JDBC-ODBC 桥驱动程序：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2. 创建数据库连接

在加载 JDBC 驱动程序之后，java.sql.DriverManger 类开始接管驱动程序，识别出可以使用的驱动程序。DriverManager 通过特定格式的字符串或 URL 与数据库系统建立连接，通过 DriverManager.getConnection()方法可以获取实现了 Connection 接口的对象。如下面的代码表示通过 JDBC-ODBC 桥建立一个访问 SQL Server 数据库的连接：

```
Connection con=DriverManager.getConnection("jdbc:odbc:local");
```

或者：

```
Connection con=DriverManager.getConnection("jdbc:odbc:local","sa","");
```


可以看出上述两个语句的区别在于：第 2 个语句包括了用户名和密码，在数据源中已经指定用户名和密码的情况下可以使用第 1 种方式。

不同的 JDBC 驱动程序的字符串或 URL 表示方式也各不相同，在使用之前需要查询相关的文档。

DriverManager 类用于管理数据库驱动程序，作用于用户和驱动程序之间。它跟踪可用的驱动程序，并在数据库和相应的驱动程序之间建立连接，也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等事务。DriverManager 类的主要成员方法如表 12-2 所示。

表 12-2　DriverManager 主要成员方法

方　法	含　义
static void deregisterDriver(Driver driver)	从数据库驱动程序列表中删除数据库驱动程序
static Connection getConnection(String url)	创建数据库连接
static Connection getConnection(String url, Properties info)	创建数据库连接(使用属性信息)
static Connection getConnection(String url, Strng username, String password)	创建数据库连接(使用用户名、密码)
static Driver getDriver(String url)	获取数据库驱动程序
static Enumueration getDrivers()	获取数据库驱动程序列表
static int getLoginTimeout()	获取数据库连接登录超时
static void setLoginTimeout(int timeout)	设置数据库连接超时
static PrintWriter getLogWriter()	获取日志记录器
static void setLogWriter(PrintWriter out)	设置日志记录器
static void println(String message)	向日志记录器中写入消息
static void registerDriver(Driver driver)	在 DriverManager 中注册驱动程序

Connection 用来表示数据库的连接对象，对数据库的一切操作都建立在 Connection 基础之



上。Connection 的主要成员方法如表 12-3 所示。

表 12-3　Connection 主要成员方法

方　法	含　义
void clearWarnings()	清除连接的所有警告信息
void close()	释放连接对象的数据库和 JDBC 资源
void commit()	提交对数据库的改动
Statement createStatement()	创建一个 statement 对象
Statement createStatement(int resultSetType, int resultSetConcurrency)	创建一个 statement 对象，它将生成具有特定类型和并发性的结果集
String getCatalog()	获取连接对象的当前目录名
DatabaseMetaData getMetaData()	获取连接对象的数据库元数据
boolean isClosed()	判断连接是否已关闭
boolean isReadOnly()	判断连接是否为只读模式
CallableStatement prepareCall(String sql)	创建 CallableStatement
PreparedStatement prepareStatement(String sql)	创建 PreparedStatement
void rollback()	回滚当前事务中的所有改动
void setReadOnly()	设置连接的只读模式



3. 建立和执行数据处理语句

建立了数据库连接之后，就可以对数据库中的数据进行操作了。在第 11 章中已经介绍了，数据操作语言包括查询、插入、更新和删除数据库表中的数据。利用 JDBC 访问数据库传送 SQL 命令需要使用 Statement 接口的实现类。调用 Connection 的 createStatement 方法新建一个 Statement 对象，然后使用这个对象执行所需的 SQL 语句即可。创建 Statement 对象的语句如下：

```
Statement stmt=con.createStatement();
```

Statement 对象的 executeQuery 方法用于执行可以返回数据的查询操作，即 SELECT 语句。executeQuery 方法返回包含满足条件的所有数据的结果集(ResultSet)。如下面的语句试图从数据库中查询 authors 表中的所有字段的数据：

```
String queryStr="SELECT * FROM authors";
ResultSet rs=stmt.executeQuery(queryStr);
```

Statement 对象的 executeUpdate 方法用于执行包括插入(INSERT)、更新(UPDATE)和删除(DELETE)等对数据进行修改的 SQL 命令。executeUpdate 方法返回一个整数，表示 Statement 修改的行数，如下面的代码正确执行后 count 被赋值为 1。

```
int count = stmt.executeUpdate("INSERT into authors
values('123-45-6789','Test','Test','123 456-7890', '123 St.', 'Dublin','CA','95689',1)")
```

Statement 对象也提供了一个 execute 方法来执行通用 SQL 命令，execute 方法返回一个布尔值。如果执行 SQL 命令返回一个或多个 ResultSet 对象，则 execute 方法返回 true；如果执行 SQL 命令返回一个更新计数或者没有任何结果，则 execute 方法返回 false。可以使用 Statement 的 getResultSet 或 getUpdateCount 方法来获取执行结果(ResultSet 或者更新计数)。

事实上 JDBC API 中拥有 3 种 Statement 对象：Statement、PreparedStatement(继承自 Statement)和 CallableStatement(继承自 PreparedStatement)。它们作为在连接上执行 SQL 语句的容器，用于发送特定类型的 SQL 语句：Statement 对象用于执行不带参数的 SQL 语句，提供了执行语句和获取结果的基本方法；PreparedStatement 对象用于执行带或不带参数的预编译 SQL 语句，添加了处理 IN 参数的方法；CallableStatement 对象用于执行对数据库的存储过程的调用，添加了处理 OUT 参数的方法。

PreparedStatement 在 SQL 字符串中使用问号(?)作为未知参数的占位符，这些参数在运行时由应用程序指定。使用 PreparedStatement 对象时首先要用 Connection 对象的 prepareStatement 方法创建 PreparedStatement 对象，然后使用 setXXX()方法设置参数值，接下来就可以像处理 Statement 对象一样调用执行方法来处理了。下面的一段代码演示了如何使用 PreparedStatemet 对象。

```
String template = "UPDATE authors SET city='Beijing' where author_id= ?";
PreparedStatement ps=con.prepareStatement(template);
ps.SetString(1, "123 456-7890");
ps.executeUpdate();
```

CallableStatement 继承自 PreparedStatement，所以它的用法和 PreparedStatement 基本相似，不过需要使用 Connection 对象的 prepareCall 方法创建。CallableStatement 主要用于存储过程的调用。调用存储过程的语法格式如下：

```
{call procedure_name}                  // 过程不需要参数
{call procedure_name[(?,?,?,…)]}       // 过程需要若干个参数
{? = call procedure_name[(?,?,?,…)]}   // 过程需要若干个参数并返回一个参数
```

如果调用存储过程时需要返回结果，就需要调用 CallableStatement 对象的 registerOutParameter 方法设置输出参数，并在执行 SQL 语句后使用 getXXX 方法读取结果。下面的代码演示了如何使用 CallableStatement 对象调用存储过程(假设有一个查询指定 author 所在州 state 的存储过程 getAuthorsState)并返回结果。

```
CallableStatement cs=con.prepareCall("{call getAuthorsState (?)}");
cs.setString(1, "123 456-7890");
cs.registerOutParameter(1, java.sql.Types.CHAR);
cs.executeUpdate()
String state=cs.getString();
```

表 12-4~表 12-6 分别列出了 Statement、PreparedStatement 和 CallableStatement 的常用方法。



表 12-4 Statement 的主要成员方法

方 法	含 义
void addBatch(String sql)	在 Statement 语句中添加 SQL 批处理语句
void cancel()	取消 Statement 中的 SQL 语句指定的数据库操作命令
void clearBatch()	清除 Statement 中的 SQL 批处理语句
void clearWarnings()	清除 Statement 语句中的操作引起的警告
void close()	关闭 Statement 语句指定的数据库连接
boolean execute(String sql)	执行 SQL 语句
int[] executeBatch()	执行批处理 SQL 语句
ResultSet executeQuery(String sql)	执行数据库查询，返回结果集
int executeUpdate(String sql)	执行数据库更新
Connection getConnection()	获取对数据库的连接
int getFetchDirection()	获取从数据库表中获取行数据的方向
int getFetchSize()	获取返回的数据库结果集的行数
int getMaxFieldSize()	获取返回的数据库结果集的最大字段数
int getMaxRows()	获取返回的数据库结果集的最大行数
boolean getMoreResults()	获取 Statement 的下一个结果
int getQueryTimeout()	获取查询超时设置
ResultSet getResultSet()	获取结果集
int getUpdateCount()	获取更新记录的数量
void setCursorName(String name)	设置数据库 Cursor 的名称
void setFetchDirection(int dir)	设置数据库表中获取行数据的方向
void setFetchSize(int rows)	设置返回的数据库结果集行数
void setMaxFieldSize(int max)	设置最大字段数
void setMaxRows(int max)	设置最大行数
void setQueryTimeout(int seconds)	设置查询超时时间





表 12-5 PreparedStatement 的主要成员方法

方 法	含 义
void clearparameters ()	清除 PreparedStatement 中设置的参数
ResultSetMetaData getMetaData()	执行数据库查询，获取数据库元数据
void setArray(int index,Array x)	设置为数组类型
void setAsciiStream(int index,InputStream stream,int length)	设置为 ASCII 输入流
void setBigDecimal(int index,BigDecimal x)	设置为十进制长类型
void setBinaryStream(int index, InputStream stream,int length)	设置为二进制输入流
void setCharacterStream(int index, InputStream stream,int length)	设置为字符输入流



(续表)

方　法	含　义
void setBoolean(int index, boolean x)	设置为布尔类型
void setByte(int index,byte b)	设置为字节类型
void setBytes(int byte[] b)	设置为字节数组类型
void setDate(int index,Date x)	设置为日期类型
void setFloat(int index,float x)	设置为浮点类型
void setInt(int index,int x)	设置为整数类型
void setLong(int index,long x)	设置为长整数类型
void setRef(int index,int ref)	设置为引用类型
void setShort(int index,short x)	设置为短整数类型
void setString(int index,String x)	设置为字符串类型
void setTime(int index,Time x)	设置为时间类型



表 12-6　CallableStatement 主要成员方法

方　法	含　义
Array getArray(int i)	获取数组
BigDecimal getBigDecimal(int index) BigDecimal getBigDecimal(int index，int scale) BigDecimal getBigDecimal(String name)	获取十进制小数 (以下的所有 get 方法均具有分别使用 index 或参数名 name 作为参数的方法，如无特殊情况，只列出使用 index 一种方法)
boolean getBoolean(int index)	获取逻辑类型
byte getByte(int index)	获取字节类型
Date getDate(int index) Date getDate(int index,Calendar cal)	获取日期类型
double getDouble(int index)	获取双精度类型
float getFloat(int index)	获取浮点类型
int getInt(int index)	获取整数类型
long getLong(int index)	获取长整数类型
Object getObject(int index) Object getObject(int index,Map map)	获取对象类型
Ref getRef(int i)	获取 Ref 类型
short getShort(int index)	获取短整数类型
String getString(int index)	获取字符串类型
Time getTime(int index) Time getTime(int index,Calendar cal)	获取时间类型
void registerOutputParameter(int index) void registerOutputParameter(int index,int type) void registerOutputParameter(int index,int type,int scale)	注册输出参数



4. 处理所得结果

ResultSet 对象是一个与查询结果相对应的数据行集合。ResultSet 使用内置的游标来跟踪当前行，开发人员可以非常方便快捷地读取每行的数据。调用 ResultSet 的 next 方法即可将游标移到下一行。ResultSet 提供了多个 get 方法来读取当前行的各个字段值，如 getInt 方法获取整形值(int)，getString 方法则获取字符串型数据。每种类型 get 方法都有两个重载方法，一个使用代表列索引的 int 型参数，另一个则使用字段名或别名的 String 型参数。ResultSet 对象也具有一些更新的方法以对数据库进行修改，相关方法请自行查阅 Java 相关文档。下面的代码是通过循环读取 ResultSet 对象 rs 的每一行数据并显示出来：

```
while(rs.next()) {
    System.out.println(rs.getString("au_fname"));
    System.out.println(rs.getString("phone"));
}
```

表 12-7 列出了 ResultSet 对象的常用成员方法。



表 12-7　ResultSet 的主要成员方法

方　法	含　义
boolean absolute(int row)	将指针移动到结果集对象的某一行
void afterLast()	将指针移动到结果集对象的末尾
void beforeFirst()	将指针移动到结果集对象的头部
boolean first()	将指针移动到结果集对象的第一行
Array getArray(int row)	获取结果集中的某一行并将其存入一个数组
boolean getBoolean(int columnIndex) boolean getBoolean(String columnName)	获取当前行中某一列的值，返回一个布尔值 (以下的 get 方法都具有两个重载方法，这里略过)
byte getByte(int columnIndex)	获取当前行中某一列的值，返回一个字节型值
short getShort(int columnIndex)	获取当前行中某一列的值，返回一个短整型值
int getInt(int columnIndex)	获取当前行中某一列的值，返回一个整型值
long getLong(int columnIndex)	获取当前行中某一列的值，返回一个长整型值
double getDouble(int columnIndex)	获取当前行中某一列的值，返回一个双精度型值
float getFloat(int columnIndex)	获取当前行中某一列的值，返回一个浮点型值
String getString(int columnIndex)	获取当前行中某一列的值，返回一个字符串
Date getDate(int columnIndex)	获取当前行中某一列的值，返回一个日期型值
Object getObject(int columnIndex)	获取当前行中某一列的值，返回一个对象
Statement getStatement()	获得产生该结果集的 Statement 对象
URL getURL(int columnIndex)	获取当前行中某一列的值，返回一个 java.net.URL 型值
boolean isBeforeFirst()	判断指针是否在结果集的头部
boolean isAfterLast()	判断指针是否在结果集的末尾

(续表)

方　法	含　义
boolean isFirst()	判断指针是否在结果集的第一行
boolean isLast()	判断指针是否在结果集的最后一行
boolean last()	将指针移动到结果集的最后一行
boolean next()	将指针移动到当前行的下一行
boolean previous()	将指针移动到当前行的上一行

5. 关闭数据库连接

在处理完所有数据库操作之后需要及时关闭数据库连接，以释放所占用的数据库资源。关闭数据库连接使用相应的 close 方法，ResultSet，Statement 以及 Connection 对象都提供了 close 方法。需要注意的是，在关闭数据库连接的时候要以创建相关对象相反的顺序进行关闭，例如，如果关闭 Statement，则 ResultSet 中的所有数据也会被关闭。下面的代码演示了按顺序关闭结果集，语句和数据库连接的过程：

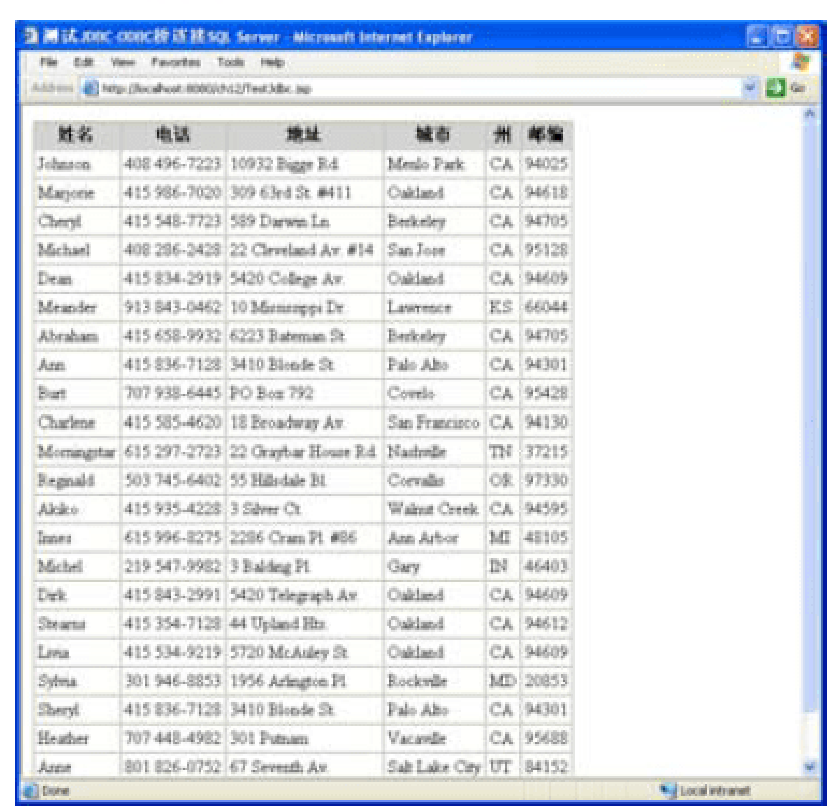
```
if(rs!=null) rs.close();
if(stmt!=null) stmt.close();
if(con!=null) con.close();
```

下面的代码是在 JSP 中通过 JDBC-ODBC 桥访问 SQL Server 附带的 pubs 数据库中 authors 表的实例，综合应用了上面介绍的各个实现步骤和方法：

```
//TestJdbc.jsp
<html>
<head>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>测试 JDBC-ODBC 桥连接 SQL Server</title>
</head>
<body>
<%
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con=DriverManager.getConnection("jdbc:odbc:local");
  Statement stmt=con.createStatement();
  String queryStr="SELECT * FROM authors";
  ResultSet rs=stmt.executeQuery(queryStr);
%>
```

```
<table border=1 cellspacing="0" cellpadding="3">
 <tr bgcolor="lightgrey">
   <th align="center">姓名</th>
   <th align="center">电话</th>
   <th align="center">地址</th>
   <th align="center">城市</th>
   <th align="center">州</th>
   <th align="center">邮编</th>
 </tr>
<%
 while(rs.next()) {
%>
 <tr>
   <td><%=rs.getString("au_fname")%></td>
   <td><%=rs.getString("phone") %></td>
   <td><%=rs.getString("address") %></td>
   <td><%=rs.getString("city") %></td>
   <td><%=rs.getString("state") %></td>
   <td><%=rs.getString("zip") %></td>
 </tr>
<%
 }
 if(rs!=null) rs.close();
 if(stmt!=null) stmt.close();
 if(con!=null) con.close();
%>
</table>
</body>
</html>
```

运行此页面，结果如图 12-10 所示。



姓名	电话	地址	城市	州	邮编
Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025
Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618
Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705
Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128
Dean	415 834-2919	5420 College Av.	Oakland	CA	94609
Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	66044
Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705
Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301
Burt	707 938-6445	PO Box 792	Covelo	CA	95428
Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130
Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215
Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	97330
Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595
Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	48105
Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403
Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609
Stearns	415 354-7128	44 Upland Hts.	Oakland	CA	94612
Livia	415 534-9219	5720 McAuley St.	Oakland	CA	94609
Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	20853
Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301
Heather	707 448-4982	301 Putnam	Vacaville	CA	95688
Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152

图 12-10　通过 JDBC-ODBC 桥查询数据库

12.3 JDBC 数据类型

各个数据库厂商都有各自不同的数据类型定义，而 Java 有一套标准的数据类型定义，如何才能将这两者结合起来，准确无误地在二者之间进行转换，这就是 JDBC 需要考虑的。JDBC 根据常用的 SQL 数据类型定义了一组数据类型，并在 JDBC API 中将这些类型和 Java 数据类型进行转化，使开发人员不用过多地考虑底层的数据细节。

所有 JDBC 数据类型都在 java.sql.Types 类中作了详细的规定，表 12-8 给出了 JDBC 中的常用数据类型和 Java 数据类型之间的对应关系。

表 12-8　SQL 数据类型和 Java 数据类型的对应关系

SQL 数据类型	Java 数据类型
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

12.4 数据库事务

事务是数据操作的基本逻辑单位。一个事务可能包括多个 SQL 语句，它们作为一个单独的



元素得到执行。当事务执行过程中发生错误时，需要有一种方法使数据库忽略当前状态，并回到事务执行前的程序状态。这两种情况在数据库术语中分别称为提交事务和回滚事务。为了处理这两种情况，JDBC API 的 Connection 对象包括了两个方法 commit()和 rollback()，分别用于实现事务的提交和回滚。在使用这两个方法时通常要使用 try ... catch 语句捕获数据库实际运行操作时可能发生的 SQLException 异常。

不同的数据库系统采用不同的事务处理机制，JDBC API 支持不同类型的事务，它们由 Connection 对象的 setTransactionLevel 方法指定。在 JDBC API 中可以获得的事务级别如表 12-9 所示。

表 12-9　事 务 级 别

事 务 级 别	含　义
TRANSACTION_NONE	不支持事务
TRANSACTION_READ_UNCOMMITTED	一个事务在提交前其变化对于其他事务来说是可见的。 由于数据处于未提交状态，如果另外的事务请求回滚其修改的数据，从而导致前一事务数据损坏，这称为脏读； 如果事务获取数据，随后另一事务修改该数据，则原有事务第二次获取该数据就会发生不一致，这称为不可重复读； 如果一个事务通过某种查询方式获取数据，另一事务修改数据的一部分，那么原有事务可能无法再次获取该数据，这称为虚读
TRANSACTION_READ_COMMITTED	事务不能读取其他事务未提交的数据，仍然允许不可重复读和虚读
TRANSACTION_REPEATABLE_READ	事务保证能够再次读取相同的数据而不会失败，但虚读仍然会出现
TRANSACTION_SERIALIZABLE	最高的事务级别，它防止脏读、不可重复读和虚读



TRANSACTION_SERIALIZABLE 级别下的事务可以保证最高程度的数据完整性，但事务保护是以牺牲性能为代价的，保护级别越高，性能损失也就越大。

设置 Connection 对象 con 的事务级别的方法如下：

```
con.setTransactionLevel(TRANSACTION_SERIALIZABLE) ;
```

也可以使用 getTransactionLevel()方法获取当前事务的级别，如下：

```
con.getTransactionLevel();
```

默认情况下，JDBC 驱动程序运行在【自动提交】模式下，即发送到数据库的所有命令运行在它们自己的事务中。这样做虽然方便，但付出的代价是程序运行时的开销比较大。可以利用批处理操作来减小这种开销，因为在一次批处理操作中可以执行多个数据库更新操作。但批处理操作要求事务不能处于自动提交模式下。为此，要首先禁用自动提交模式：

```
con.setAutoCommit(false);
```

下面是一个批处理操作的例子：

```
Try {
Statement stmt = con.createStatement() ;
stmt.addBatch("INSERT into authors values('123-45-6789','Test1','Test1','123 456-7890', '123 St.',
'Dublin','CA','95689',1) ");
stmt.addBatch("INSERT into authors values('234-56-7890','Test2','Test2','123 456-7890', '123 St.',
'Dublin','CA','95689',1) ");
stmt.addBatch("INSERT into authors values('345-67-8901','Test3','Test3','123 456-7890', '123 St.',
'Dublin','CA','95689',1) ");
int[] updateCounts = stmt.executeBatch() ;
con.commit() ;
}
catch (BatchUpdateException batchEx) {
con.rollback()
}
```


executeBatch()方法返回一个更新计数的数组，每个值对应于批处理操作的一个命令。批处理操作可能会抛出一个类型为 BatchUpdateException 的异常，该异常表明批处理操作中至少有一条命令失败了。

12.5 上机练习

本章上机实验主要练习如何使用 JDBC 技术进行数据库编程。其中重点掌握如何配置 ODBC、如何加载数据库驱动程序、如何建立连接、如何创建数据库 SQL 命令访问和处理数据库数据以及如何处理数据库事务等。

下面以在 JSP 页面中应用 JDBC 技术查询数据库中的数据为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

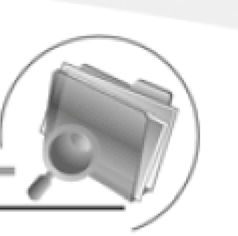
(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击 Finish 按钮后 Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中将会出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Other】命令，弹出 New 对话框以创建 JSP 文件。



(7) 在 New 对话框中选择【Web】|【JSP】选项，单击【Next】按钮弹出 New JavaServer Page 对话框。

(8) 在 New JavaServer Page 对话框中的 File name 文本框中输入文件名，如 TestJdbc.jsp。

(9) 单击【Finish】按钮新建 JSP 页面文件。

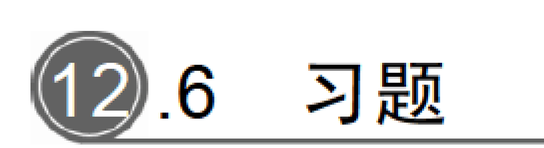
(10) Eclipse 会创建一个包括了基本的页面框架的新文件。在编辑窗口中输入 TestJdbc.jsp 页面代码即可。

(11) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(12) 根据错误提示，修改所有可能的语法错误。

(13) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(14) 打开 Internet Explorer 浏览器，输入对应的 URL，如：http://localhost:8080/testTomcat/TestJdbc.jsp。观察页面运行效果。



12.6　习题

12.6.1　填空题

1. JDBC 数据库驱动程序包括__________、__________、__________和__________。

2. JDBC 可选包 API 包括__________、__________、__________和__________等包的内容。

12.6.2　选择题

1. 以下几种事务中，级别最低的是(　　　　)。

A. TRANSACTION_READ_UNCOMMITTED

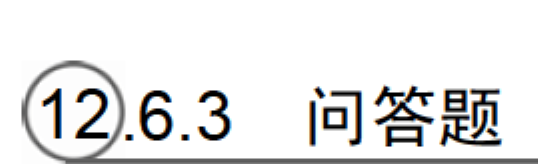
B. TRANSACTION_READ_COMMITTED

C. TRANSACTION_REPEATABLE_READ

D. TRANSACTION_SERIALIZABLE

2. 以下数据类型中，哪个不是 JDBC 支持的类型(　　　　)。

A. VARCHAR　　B. BLOB　　C. DATE　　D. TIME



12.6.3 问答题

简述采用 JDBC 访问数据库的步骤。



第13章 JSP 与 XML

学习目标

XML(可扩展标记语言)的发展是近几年来计算方式的一大变革。XML 在存储和转化数据方面具有很大的优势，XML 本质上是一种纯文本格式，因此它可以很容易地在不同平台、操作系统和应用程序之间传输。

XML 作为业界的一种标准已经得到了广泛应用，不管是.NET 平台还是 J2EE 平台，甚至一些更老的应用平台都对 XML 提供了强大的支持。在 JSP 应用中，XML 已经成为必不可少的组成部分。JSP 应用使用的配置文件 server.xml 和 web.xml 等本身就是 XML 文件。

本章重点

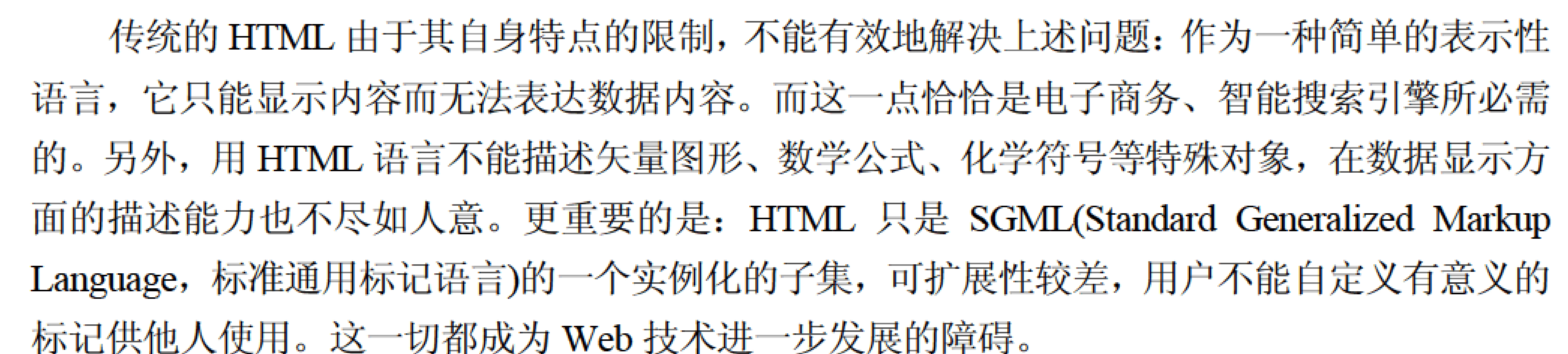
- XML 的语法结构
- DTD 和 Schema
- 使用 DOM 操作 XML 文件
- 使用 SAX 操作 XML 文件

13.1 XML 简介

XML 的全称为 eXtensible Markup Language(可扩展标记语言)，它是由业界的领导厂商组成的 W3C 委员会制定的一种标准，目的是加强网络应用的功能，提供统一的网络应用数据传输和存取协议。XML 提供了一种崭新的网络数据处理方式，在任何平台、任何语言环境下均能通行。

13.1.1 XML 与 HTML

Internet 提供了全球范围的网络互联与通信，Web 技术的发展更是一日千里，其丰富的信息资源给人们的学习和生活带来了极大的便利。特别是应运而生的 HTML(超文本标记语言)，以其简单易学、灵活通用的特性，使得人们发布、检索、交流信息都变得非常简单。然而，电子商务、电子出版、远程教育等基于 Web 的新兴领域的全面兴起使得传统的 Web 资源更加复杂化、多样化，数据量的日趋增大对网络的传输也提出了更高的要求。同时，人们对 Web 服务功能的需求也达到更高的标准，比如：用户需要对 Web 资源进行智能化的语义搜索、对数据按照不同的需求进行多样化显示等个性化服务；公司和企业要为客户创建和分发大量有价值的文档信息，以降低生产成本，以及对不同平台、不同格式的数据源进行数据集成和数据转化等，这些需求都变得越来越广泛和迫切。



传统的 HTML 由于其自身特点的限制，不能有效地解决上述问题：作为一种简单的表示性语言，它只能显示内容而无法表达数据内容。而这一点恰恰是电子商务、智能搜索引擎所必需的。另外，用 HTML 语言不能描述矢量图形、数学公式、化学符号等特殊对象，在数据显示方面的描述能力也不尽如人意。更重要的是：HTML 只是 SGML(Standard Generalized Markup Language，标准通用标记语言)的一个实例化的子集，可扩展性较差，用户不能自定义有意义的标记供他人使用。这一切都成为 Web 技术进一步发展的障碍。

SGML 是一种通用的文档结构描述标记语言，为语法标记提供了异常强大的工具，而且具有极好的扩展性，因此在数据分类和索引方面非常有用。但 SGML 复杂度太高，不适合网络的日常应用，加上开发成本高、不被主流浏览器所支持等原因，使得 SGML 的推广受到阻碍。在这种情况下，开发一种兼具 SGML 的强大功能、可扩展性，同时又具有 HTML 的简单性的语言势在必行。于是诞生了 XML 语言。

XML(eXtensible Markup Language，可扩展标记语言)是由 W3C 于 1998 年 2 月发布的一种标准。它同样是 SGML 的一个简化子集，它将 SGML 的丰富功能与 HTML 的易用性结合到 Web 应用中，以一种开放的自我描述方式定义数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系。这样所组织的数据对于应用程序和用户来说都是友好的、可操作的。

XML 的优势之一就是它允许各个组织、个人建立适合自己需要的标记集合，并且将这些标记迅速地投入使用。这一特征使得 XML 可以在电子商务、政府文档、司法、出版、CAD/CAM、保险机构、厂商和中介组织信息交换等领域中一展身手，针对不同的系统、厂商提供各具特色的独立解决方案。

XML 的最大优点在于它的数据存储格式不受显示格式的制约。一般来说，一个文档包括 3 个要素：数据、结构以及显示方式。对于 HTML 来说，显示方式内嵌在数据中，这样，在创建文本时，就要时时考虑输出格式。如果因为需求不同而需要对同样的内容进行不同风格的显示，就要从头创建一个全新的文档，重复工作量很大。此外 HTML 缺乏对数据结构的描述，对于应用程序理解文档内容、抽取语义信息都有诸多不便。



XML 把文档的 3 要素独立开来，分别处理。首先把显示格式从数据内容中独立出来，保存在样式表单文件(Style Sheet)中，这样如果需要改变文档的显示方式，只需修改样式表单文件即可。XML 的自我描述性质能够很好地表现许多复杂的数据关系，使得基于 XML 的应用程序可以在 XML 文件中准确高效地搜索相关的数据内容，而忽略其他不相关部分。XML 还有许多其他优点，比如，它有利于不同系统之间的信息交流，完全可以充当网际语言，并有希望成为数据和文档交换的标准机制。

但是，XML 并不能完全取代 HTML，毕竟 HTML 是最为方便、快捷的网上信息发布方式。而且 HTML 是描述数据显示的语言，而 XML 是描述数据及其结构的语言，二者在功能上也是截然不同的。表 13-1 给出了两者的比较。

表 13-1　HTML 和 XML 的对比

	HTML	XML
可扩展性	不具有扩展性	是源标记语言，可用于定义新的标记语言
侧重点	侧重于如何表现信息	侧重于如何结构化地描述信息
语法要求	不要求标记的嵌套、配对等，不要求标记之间具有一定的顺序	严格要求嵌套、配对和遵循 DTD 的树形结构
可读性及可维护性	难于阅读、维护	结构清晰，便于阅读、维护
数据和显示的关系	内容描述与显示方式整合为一体	内容描述与显示方式相分离
保值性	不具有保值性	具有保值性



13.1.2　XML 基本语法

首先介绍 XML 文档的基本单元：元素，它的语法格式如下：

```
<tag>Element</tag>
```

元素由起始标签、元素内容和结束标签组成。用户把要描述的数据对象放在起始标签和结束标签之间。例如：

```
<name>张三</name>
```

无论文本内容有多长或者多么复杂，都能用 XML 来描述。XML 元素中可以嵌套别的元素，这样可以使相关信息构成等级结构。

【例 13-1】 XML 基本语法。

<books>元素包括了所有书的信息，每本书都由<book>元素来描述，而<book>元素中又嵌套了<isbn>、<author>、<title>和<price>元素。

```
//book.xml
<?xml version="1.0" encoding="utf-8" ?>
```

```
<?xml-stylesheet type="text/xsl" ?>
<books xmlns="http://tempuri.org/book.xsd">
  <book>
      <isbn>1</isbn>
      <author>金庸</author>
      <title>射雕英雄传</title>
      <price currency="RMB">90</price>
  </book>
  <book>
      <isbn>2</isbn>
      <author>古龙</author>
      <title>多情剑客无情剑</title>
       <price>70</price>
 </book>
 <book>
       <isbn>3</isbn>
      <author>金庸</author>
      <title>鹿鼎记</title>
       <price>150</price>
 </book>
 <book>
       <isbn>4</isbn>
       <author>古龙</author>
      <title>萧十一郎</title>
       <price>65</price>
 </book>
 <book>
     <isbn>5</isbn>
     <author>温瑞安</author>
     <title>四大名捕</title>
     <price>15</price>
 </book>
</books>
```

除了元素，XML 文档中能出现的有效对象包括处理指令、注释、根元素、子元素和属性。

⊙ 处理指令

处理指令给 XML 解析器提供信息，使其能够正确解释文档内容，它的起始标识是<?，结束标识是?>。常见的 XML 声明就是一个处理指令如下：

```
<?xml version="1.0" encoding="utf-8" ?>
```





处理指令还可以有其他用途，比如把一个样式表单文件应用到 XML 文档中用以显示。如【例 13-1】中的语句：

```
<?xml-stylesheet type="text/xsl" ?>
```

◉ 注释

XML 中的注释使用<!--和-->作为起始和结束标记，可以出现在 XML 元素间的任何地方，但是不可以嵌套：

```
<!--这是一个注释-->
```

◉ 根元素和子元素

如果一个元素从文件头的序言部分之后开始一直到文件结尾，包含了文件中所有的数据信息，就称之为根元素。如【例 13-1】中的<books>元素。

XML 元素是可以嵌套的，被嵌套在内的元素称为子元素。在【例 13-1】中，<book>就是<books>的子元素。



◉ 属性

属性给元素提供进一步的说明信息，它必须出现在起始标签中。属性以名称/值对出现，属性名不能重复，名称与取值之间用等号(=)分隔，并用引号把取值引起来。例如【例 13-1】中的属性 currency 说明了价格的货币单位是人民币：

```
<price currency="RMB">90</price>
```

XML 文档的基本结构由序言部分和一个根元素组成。XML 文档的序言部分包括了 XML 声明和 DTD(或者是 XML Schema)，DTD(Document Type Define，文档定义类型)和 XML Schema 都是用来描述 XML 文档结构的，也就是描述元素和属性是如何联系在一起的。例如，在【例 13-1】中的文档前面的序言部分，包括了 XML 声明，在根元素<books>中使用的 book.xsd 命名空间中的 XML Schema。后面将具体介绍 XML Schema 和 DTD。

一个 XML 文档中有且仅有一个根元素，其他的所有元素都是它的子元素，在【例 13-1】中，<books>就是根元素。

一个 XML 文档首先应当是【格式良好的】(Well-Formed)，【格式良好的】XML 文档除了要满足根元素惟一的特性之外，还应该包括：

(1) 起始标签和结束标签应当匹配：结束标签是必不可少的。

(2) 大小写要一致：XML 对字母的大小写是敏感的，<book>和<Book>在 XML 中是完全不同的两个标签，所以结束标签在匹配时一定要注意大小写一致。

(3) 元素应当正确嵌套：子元素应该完全包括在父元素中，如下所示的就是嵌套错误：

```
<A> <B> </A> </B>
```

正确的嵌套方式如下：

```
<A> <B> </B> </A>
```

(4) 属性必须包括在引号中。

(5) 元素中的属性是不允许重复的。

XML 文档的【有效性】是指一个 XML 文档应当遵守 DTD 文件或是 Schema 的规定，【有效的】XML 文档肯定是【格式良好的】。

不同的 XML 文档中很可能会定义许多名字相同而意义不同的元素或属性，尤其在把不同的 XML 文档合而为一时，更容易产生冲突。名称空间就是为了解决这个问题而提出的，它用 URI(Uniform Resource Indicator，统一资源指示器)加以区别，是在 XML 文件的元素和属性中出现的所有名称的集合。【例 13-1】中 books 元素后即有属性 xmlns 指定了其使用的命名空间：

```
<books xmlns="http://tempuri.org/book.xsd">
```

有了名称空间，用户就可以保证在文件中使用的名称是惟一的。对元素的 xmlns 属性进行定义，就表示对该元素指定了一个名称空间。namespace_name 必须是一个有效的 URI。

如果对一个元素定义了默认名称空间，那么该元素及其子元素，包括它们的属性都会自动成为该名称空间的一部分，而不用在每一个元素和属性前面一一标明了。



13.1.3　DTD 与 Schema

DTD(文档类型定义)是用来描述 XML 文件的逻辑结构的一种语言。它最大的作用在于验证 XML 文件逻辑结构的正确性。如果一个 XML 使用 DTD 来描述，那么当 XML 解析器解析 XML 文件时就会调用 DTD 来验证文件是否与事先定义的格式一致，从而减少程序员必须手工进行的错误处理。

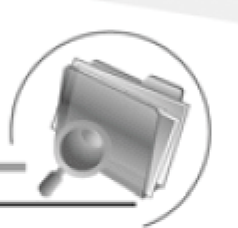
例如，针对【例 13-1】的 book.xml，可以定义下面的 DTD 文件：

```
//books.dtd
<!ELEMENT books (book)*>
<!ELEMENT book (isbn, author, title, price)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency CDATA #IMPLIED>
```

DTD 对 XML 文件的逻辑结构作了严格规定。首先，对于 books 元素，它是由任意多个 book 元素构成。而 book 元素则按顺序必须包括 isbn、author、title 和 price 这 4 个子元素。4 个子元素均使用 String 类型描述，而 price 元素另外定义了一个称为 currency 的属性。

为了将 DTD 与 XML 文件关联起来，需要在 XML 文件的开头部分加入以下说明：

```
//booksWithDTD.xml
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE books SYSTEM "books.dtd">
<books> ……
```



Eclipse 开发工具提供了非常好的 DTD 验证机制。如果在 Eclipse 中对上述的 booksWithDTD.xml 或 books.dtd 文件单独进行简单的修改，重新打开 booksWithDTD.xml 就会显示相应的错误警告信息，如图 13-1 所示。

```
<?xml version="1.0" encoding="utf-8" ?>

<!DOCTYPE books SYSTEM "books.dtd">
<?xml-stylesheet type="text/xsl" ?>          The content of element type "books" must match "(book)*".
<books>
    <book>
        <isbn>1</isbn>
        <author>金庸</author>
        <title>射雕英雄传</title>
        <price currency="RMB">90</price>
    </book>
    <book>
        <isbn>2</isbn>
        <author>古龙</author>
        <title>多情剑客无情剑</title>
        <price>70</price>
    </book>
    <book>
        <isbn>3</isbn>
        <author>金庸</author>
        <title>鹿鼎记</title>
        <price>150</price>
    </book>
    <book>
        <isbn>4</isbn>
```

图 13-1　DTD 验证

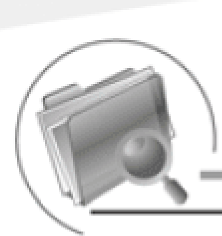


虽然 DTD 能够很好地组织 XML 文档，不过在今天的 Web 开发环境下它的局限性也变得非常明显。首先，DTD 文档本身并不是符合 XML 规范的，不能使用标准的 XML 工具对其进行编辑，在程序开发过程中也很难对 DTD 中包含的信息进行有效的利用；其次，DTD 不能有效地支持命名空间，面对复杂的 Web 应用环境 DTD 很难胜任；再次，DTD 并不支持多样化的数据类型，可以看到，在上面例子中都采用了字符串作为元素的类型，因为 DTD 的类型非常有限；最后，由于不支持自定义类型，所以 DTD 缺乏扩展性。因此，越来越多的程序员和软件公司开始舍弃 DTD 而转向 Schema。

和 DTD 一样，Schema 也是用来描述和规范 XML 文件逻辑结构的。与 DTD 不同的是，Schema 本身就是一个有效的 XML 文档，因此可以更直观地了解到 XML 的结构。同时 Schema 对命名空间的支持、内置的多种简单和复杂的数据类型以及对自定义类型的扩展都消除了 DTD 的一切局限性和弱点，已经成为目前 XML 应用的统一规范。

下面的代码是针对 books.xml 的 Schema，可以非常方便地验证 XML 文件是否符合对应的 Schema 的规范。

```
//books.xsd
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema id="books" targetNamespace="http://tempuri.org/book.xsd" xmlns="http://tempuri.org/book.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="books" >
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
```




```
            <xs:element name="book">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="isbn" type="xs:string" minOccurs="0" />
                  <xs:element name="author" type="xs:string" minOccurs="0" />
                  <xs:element name="title" type="xs:string" minOccurs="0" />
                  <xs:element name="price" nillable="true" minOccurs="0"
                   maxOccurs="unbounded" type="xs:float">
                   </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:schema>
```



Schema 使 XML 文档的结构简单明了，这里就不再赘述了。

13.2 XML 在 JSP 中的应用

在本章开头曾经提到，XML 已经成为 J2EE 平台必不可少的组成部分。在 JDK1.5 中就提供了众多的 XML 相关类，包括 javax.xml 系列包，org.w3c.dom 系列包，org.xml.sax 系列包以及 com.sun.org.apache.xalan、xerces、xml 和 xpath 系列包等。本节简单介绍一些 XML 处理器的概念。

前面已经介绍，XML 文件是一种自描述的数据结构，因此采用通用的计算机工具就能对其内容进行解析以选择相应的方式处理其中的数据。最新的 JDK1.5 中提供了 JAXP(Java API for XML Parsing)工具来实现对 XML 的解析，JAXP 主要提供了两种方式：DOM 和 SAX。

DOM(Document Object Model)是由 W3C 组织提出的对 XML 结构文档进行支持的标准规范。DOM 将 XML 文档按其内容的数据结构方式进行组织，即以节点树的方式来表示。解析器读取 XML 文档，并在内存中建立关于整个文档对象的一棵节点树，应用程序可以在树中进行遍历，编辑任意节点，做任何想做的操作。只是对于比较复杂的数据结构来说，建立节点树的空间成本比较大，另外即使应用程序只需要处理某一个简单节点也需要读取整棵节点树。

SAX(Simple API for XML)事实上是由个人发起的，虽然并没有被任何协会推荐为标准，却因为它高效而灵活的特性被广泛接受，并成为一个事实上的标准。SAX 将 XML 转换成数据流以描述 XML 文档，把每个标记的打开或关闭和每个文本块与一个事件关联起来。开发人员需要编写事件处理程序，并在处理 XML 文档过程中触发这些事件。使用 SAX 可以处理相应的事件，而不用等待整个文档的读取，也可以跳过应用程序不关心的数据，从而能够非常高效地处理 XML 数据的解析。不过当需要处理整个文档，或者随机访问文档的任意部分时，SAX 表现的就差强人意了。



13.3　使用 DOM 操作 XML 文件

本节将介绍如何使用 DOM 来操作 XML 文件。

13.3.1　一个简单的 DOM 读取 XML 节点的例子

下面的实例演示了如何使用 DOM 来读取 XML 文件中指定节点的信息。首先需要通过 DocumentBuilderFactory 类创建 DocumentBuilder 对象，然后调用该对象的 parse 方法读取 XML 文件并将其解析为 Document 对象。这里采用了【例 13-1】中使用的 book.xml 文件，寻找其中标记为 book 的节点，将其读取到一个节点列表(NodeList)对象中。通过调用 NodeList 对象的方法，将节点的信息显示在浏览器中。

```
//domRead.jsp
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="javax.xml.parsers.DocumentBuilderFactory,
                  javax.xml.parsers.DocumentBuilder,
                  org.w3c.dom.*"
%>
<%
  DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
  DocumentBuilder db=dbf.newDocumentBuilder();
  Document doc=db.parse("F:\\Work\\workspace\\testTomcat\\ch13\\book.xml");
  NodeList nl=doc.getElementsByTagName("book");
%>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>演示 DOM 读取 XML 数据</title>
</head>
<body>
<table >
<%
  for(int i=0; i<nl.getLength();i++)
  {
    NodeList nlChild=nl.item(i).getChildNodes();
%>
<tr>
```



```
<%
      for(int j=0; j<nlChild.getLength(); j++)
      {
         if(j%2!=0) {
%>
           <td width="150" nowrap>
           <%=nlChild.item(j).getNodeName()%>
           <%=nlChild.item(j).getTextContent()%><p>
<%
         }
      }
   }
%>
</table>
</body>
</html>
```

运行结果如图 13-2 所示，所有 book 子节点的名字和内容都被读取出来并显示在表格中。



图 13-2　DOM 读取 XML 文件示例

13.3.2　常用的 DOM 对象

下面重点介绍 org.w3c.dom 包中封装的一些常用 DOM 对象，并对每个对象的常用方法作了简单介绍。

1. Node(节点)

Node 对象是 DOM 结构中最为基本的对象，它代表了文档树中的一个抽象的节点。在实际



使用时，很少会真正用到 Node 对象，而是用到诸如 Element、Attr、Text 等 Node 对象的子对象来操作文档。Node 对象为这些对象提供了一个抽象的、公共的根。虽然在 Node 对象中定义了对其子节点进行存取的方法，但是有一些 Node 子对象，比如 Text 对象，它并不存在子节点。Node 对象所包含的主要方法如表 13-2 所示。

表 13-2 Node 的常用方法

方 法	描 述
getNodeName()	读取节点名
getNodeValue()	读取节点值
setNodeValue(String arg)	设置节点值
getNodeType()	读取节点类型
getParentNode()	读取父节点
getChildNodes()	读取所有的子节点(返回 NodeList)
getFirstChild()	返回第一个子节点
getLastChild()	返回最后一个子节点
getPreviousSibling()	返回当前节点的前一个兄弟节点
getNextSibling()	返回当前节点的后一个兄弟节点
getAttributes()	返回所有的属性
getOwnerDocument()	返回节点所属文档(Document)
insertBefore(Node arg1,Node arg2)	在 arg1 节点前插入 arg2 节点
replaceChilde(Node arg1, Node arg2)	将 arg1 子节点替换为 arg2 子节点
removeChild(Node arg)	删除当前节点的 arg 子节点
appendChild(Node arg)	在当前节点的最后一个子节点后附加子节点 arg
hasChildNodes()	判断当前节点是否有子节点
getNamespaceURI()	返回 Namespace 的 URI
hasAttributes()	判断当前节点是否具有属性
getBaseURI()	返回 Base 的 URI
getTextContent()	返回文本内容
setTextContent(String arg)	设置文本内容
isSameNode(Node arg)	判断是否为同一节点
isDefaultNamespace(String arg)	是否是默认 Namespace
lookupNamespaceURI(String arg)	查找 Namespace URI
isEqualNode(Node arg)	判断节点是否相等
getUserData(String arg)	返回用户定义数据
SetUserData(String arg,Object arg, UserDataHandler arg)	设置用户定义数据



2. Element(元素)

Element 对象代表的是 XML 文档中的标签元素，它继承于 Node，亦是 Node 的最主要的子对象。在标签中可以包含属性，因而 Element 对象中有存取其属性的方法，而任何 Node 中定义的方法，也可以应用于 Element 对象。表 13-3 列出了 Element 对象的常用成员方法。

表 13-3　Element 的常用方法

方　法	描　述
getAttribute(String name)	返回名为 name 的属性值
getAttributeNode(String name)	返回属性 Attr 节点
getElementsByTagName(String name)	返回标签名为 name 的所有元素的一个 NodeList
getTagName()	返回标签名
hasAttribute(String name)	判断是否具有名为 name 的属性
removeAttribute(String name)	删除名为 name 的属性
removeAttributeNode(Attr oldAttr)	删除 oldAttr 属性
setAttribute(String name, String value)	设置属性值
setAttributeNode(Attr newAttr)	设置 newAttr 属性

3. NodeList(节点列表)

顾名思义，NodeList 对象就是一个包含一个或多个 Node 对象的列表。可以简单地把它看成一个 Node 的数组，通过 item 方法来获得列表中的元素，表 13-4 列出了 NodeList 仅有的两种方法。

表 13-4　NodeList 的方法

方　法	描　述
getLength	返回 NodeList 的长度
item(int index)	返回第 index 个节点 Node

4. Document(文档)

Document 对象代表了整个 XML 文档，所有 Node 都以一定的顺序包含在 Document 对象之内，排列成一个树形的结构，开发人员可以通过遍历这棵树来得到 XML 文档的所有的内容，这也是对 XML 文档操作的起点。DOM 先通过解析 XML 源文件得到一个 Document 对象，然后再执行后续的操作。此外，Document 还包含了创建其他节点的方法，比如 createAttribut()用来创建一个 Attr 对象。它所包含的主要方法如表 13-5 所示。



表 13-5　Document 的常用方法

方　法	描　述
createAttribute(String name)	创建一个名为 name 的 Attr 对象
createCDATASection(String data)	创建一个 CDATA 节
createComment(String data)	创建注释 Comment
createDocumentFragment()	创建一个空 DocumentFragment 对象
createElement(String tagName)	创建一个标签名为 tagName 的 Element 对象
createTextNode(String data)	创建一个 Text 节点
getDoctype()	返回文档类型
getDocumentElement()	返回可以直接存取的 Document 元素
getDocumentURI()	返回 Document 的 URI
getElementById(String elementId)	返回具有 ID 属性并且其值等于 elementId 的 Element 对象
getElementsByTagName(String tagName)	返回标签名为 tagName 的 Element 对象
getInputEncoding()	返回输入编码
getXmlEncoding()	返回 XML 编码
getXmlVersion()	返回 XML 版本
importNode(Node importedNode, boolean deep)	从另一个文档导入一个节点
renameNode(Node n, String namespace URI, String qualifiedName)	重命名一个节点
setDocumentURI(String documentURI)	设置 Document URI
setXmlVersion(String xmlVersion)	设置 XML 版本



5. Attr(属性)

Attr 对象代表了某个标签中的属性。Attr 也继承于 Node，但是因为 Attr 实际上是包含在 Element 中的，所以它并不能被看作是 Element 的子对象，因而在 DOM 中 Attr 并不是 DOM 树的一部分，所以 Node 中的 getparentNode()、getpreviousSibling()和 getnextSibling()返回的都将是 null。也就是说，Attr 其实是被看作包含它的 Element 对象的一部分，它并不作为 DOM 树中的单独一个节点出现。这一点在使用时要同其他的 Node 子对象相区别。Attr 对象的主要方法如表 13-6 所示。

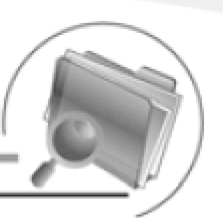
表 13-6　Attr 的常用方法

方　法	描　述
getName()	返回属性名
getOwnerElement()	返回属性所属的 Element 对象
getValue()	返回属性值
isId()	判断是否是 ID
setValue(String value)	设置属性值

13.3.3 使用 DOM 读写 XML 文档

在介绍了常用的 DOM 对象之后，就可以使用 DOM 提供的 API 对 XML 文档编写功能强大的程序。下面的代码显示了如何用 DOM 读写【例 13-1】中的 book.xml 文档。

```
//DOMReadWriteXML.java
import javax.xml.parsers.*;
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
public class DOMReadWriteXML {
 public static void main(String[] args) {
  try {
   //创建 DOM 节点树
   DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
   DocumentBuilder builder=factory.newDocumentBuilder();
   Document doc=builder.parse("book.xml");
   doc.normalize();
   String isbn="6";
   String author="梁羽生";
   String title="七剑下天山";
   String price="45";
   Text textseg;
     //创建 book 元素节点
   Element book=doc.createElement("book");
     //创建 book 元素的子节点
   Element bookisbn=doc.createElement("isbn");
   textseg=doc.createTextNode(isbn);
   bookisbn.appendChild(textseg);
   book.appendChild(bookisbn);
   Element bookauthor=doc.createElement("author");
   textseg=doc.createTextNode(author);
   bookauthor.appendChild(textseg);
   book.appendChild(bookauthor);
   Element booktitle=doc.createElement("title");
   textseg=doc.createTextNode(title);
   booktitle.appendChild(textseg);
   book.appendChild(booktitle);
   Element bookprice=doc.createElement("price");
   textseg=doc.createTextNode(price);
   bookprice.appendChild(textseg);
```



```
        book.appendChild(bookprice);
        //将 book 节点作为当前文档的 Document 元素的子节点
        doc.getDocumentElement().appendChild(book);
          //将新建的 DOM 树写入文件中
        TransformerFactory tFactory =TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer();
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new java.io.File("book.xml"));
        transformer.transform(source, result);
        //读取并显示 XML 文件内容
        NodeList books =doc.getElementsByTagName("book");
        for (int i=0;i<books.getLength();i++) {
         book=(Element) books.item(i);
         System.out.print("ISBN: ");
      System.out.println(book.getElementsByTagName("isbn").item(0).getFirstChild().getNodeValue());
         System.out.print("作者: ");
      System.out.println(book.getElementsByTagName("author").item(0).getFirstChild().getNodeValue());
         System.out.print("书名: ");
      System.out.println(book.getElementsByTagName("title").item(0).getFirstChild().getNodeValue());
         System.out.print("价格: ");
      System.out.println(book.getElementsByTagName("price").item(0).getFirstChild().getNodeValue());
         System.out.println();
        }
       }
      catch (Exception e) {
        e.printStackTrace();
       }
     }
    }
```

编译并运行该程序，结果如图 13-3 所示，DOMReadWriteXML 程序首先将新的数据写入文件中，然后通过 DOM 处理器读取并显示。

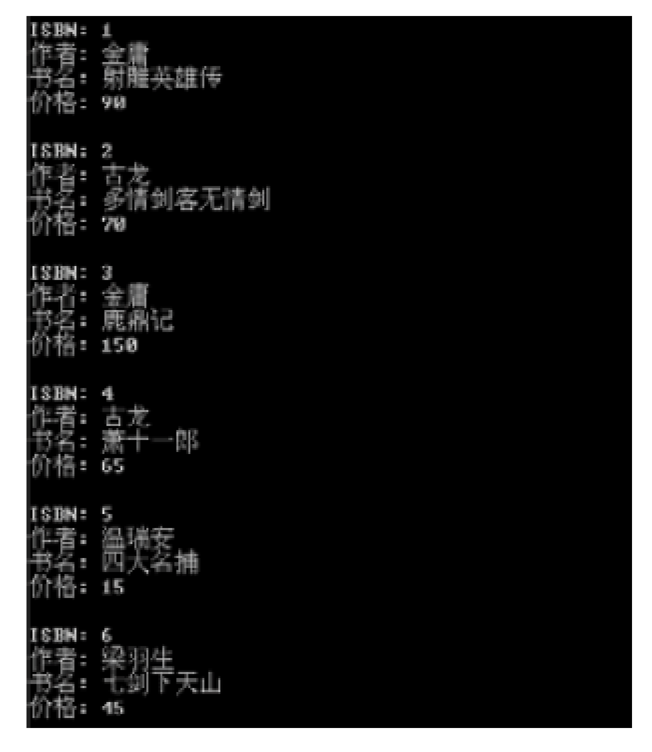


图 13-3　DOM 读写 XML 文件示例



13.4 使用 SAX 操作 XML 文件

13.2 节曾经提到，简单 XML 应用程序接口(SAX)是由个人提出的，却是 DOM 之外的另一种事实标准。和 DOM 文档驱动的重量级方法相反，SAX 是事件驱动的轻量级方法。它的作用主要是快速地读取 XML 流，并迅速地触发事件给接收对象处理。这样，即使需要处理一个非常大的 XML 数据源，也能够比较快的响应而不用构造庞大的文档对象。

13.4.1 SAX 事件处理过程

在处理 SAX 事件之前，首先需要创建一个 XMLReader 对象。XMLReader 是一个接口，一般来说都是通过 XMLReaderFactory 类的静态方法 createXMLReader 来创建一个可用的 XMLReader 对象：



```
XMLReader xmlReader= XMLReaderFactory.createXMLReader();
```

创建 XMLReader 对象之后，还需要指定合适的 SAX 事件监听程序。一般情况下，需要创建 SAX 事件监听程序对象对 XML 文档进行自定义处理。SAX 事件监听程序对象需要为不同的 SAX 事件提供恰当的接口，这些接口包括 ContentHandler、DTDHandler 和 ErrorHandler。

XMLReader 对象分别使用 setContentHandler、setDTAHandler 和 setErrorHandler 方法来设置相关的 SAX 事件处理程序。剩下的工作就是调用 XMLReader 的 parse 方法对 XML 文件源进行处理，在文件读取到相应的节点时 SAX 事件被触发，SAX 事件监听程序就会调用相应的处理程序。

13.4.2 SAX 事件处理接口

ContentHandler、DTDHandler 和 ErrorHandler 这 3 种不同的接口分别用于处理不同的事件。下面对这些接口及其主要方法分别作简要的介绍。

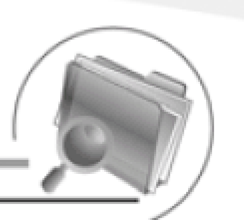
- ContentHandler

ContentHandler 用来处理 XML 内容所触发的事件，如文档或元素等。ContentHandler 是最核心的 SAX 事件处理接口，它需要实现的主要方法包括以下几种。

void characters(char[] ch, int start, int length)：当 SAX 处理器读取到 XML 文件中的字符串时触发该方法来进行处理。参数是一个字符数组以及读取的字符串在这个数组中的起始位置及长度。

startDocument()：当 SAX 处理器读取文档的开头时调用此方法。一般会在该方法中进行初始化工作。

void endDocument()：当 SAX 处理器处理文档结束时调用此方法，一般会在该方法中做一



些善后工作。

void startElement(String uri, String localName, String qName, Attributes atts)：当 SAX 处理器读到一个元素的开始标签时触发该方法。uri 参数表示其命名空间；localName 是本地名；qName 是前缀；atts 则是该标签所包含的所有属性列表。如果该 XML 文档没有使用命名空间，则 localName 和 qName 均为空(null)。

void endElement(String uri, String localName, String qName)：当 SAX 处理器读取一个元素的结束标签时调用该方法。它的参数含义与 startElement 一致。

⦿ DTDHandler

DTDHandler 用于处理与文档类型定义(DTD)有关的事件，它需实现的方法有两个。

void notationDecl(String name, String publicId, String systemId)：当 SAX 处理器读取符号描述命令时触发该方法。name 是符号名，publicId 和 systemId 分别是符号的公用和系统识别符(ID 号)。

void unparsedEntityDecl(String name, String publicId, String systemId, String notationName)：当 SAX 处理器读取不能解析的实体时调用该方法，name 表示该实体名，其他同上。



⦿ ErrorHandler

ErrorHandler 主要用于处理解析 XML 文档所出现的各种错误事件。ErrorHandler 提供了 3 个层次的错误处理，分别是警告(warning)、错误(error)和致命错误(fatal error)，对应如下 3 个方法：

```
void warning(SAXParseException exception)
void error(SAXParseException exception)
void fatalError(SAXParseException exception)
```

13.4.3　通过实例学习使用 SAX 处理 XML 文档

下面通过一个实例来学习如何使用 SAX 来处理 XML 文档，该实例继续使用【例 13-1】中的 book.xml 文件。主要目的是统计每个不同的标签出现的次数，通过在每次遇到标签时进行计数，在 XML 读取完毕后即打印出相关统计信息。在该程序中 SAXSample 类扩展了 DefaultHandler 类，DefaultHandler 是一个实现了 ContentHandler、DTDHandler 和 ErrorHandler 接口的默认处理器类。

```
import java.util.Hashtable;
import java.util.Enumeration;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class SAXSample extends DefaultHandler {
  private Hashtable tags;   //记录每个 tag 出现次数
```

```
public void startDocument() throws SAXException {
  tags = new Hashtable();
}
public void startElement(String uri, String localName,
  String qName, Attributes atts)      throws SAXException
{
  String key = localName;
  Object value = tags.get(key);
  //如果是新标签，则在 tags 中添加一条记录，否则将对应项的值加一
  if (value == null) {
    tags.put(key, new Integer(1));
  } else {      int count = ((Integer)value).intValue();
                count++;
                tags.put(key, new Integer(count));
          }
  }
public void endDocument() throws SAXException {
  Enumeration e = tags.keys();
  while (e.hasMoreElements()) {
     String tag = (String)e.nextElement();
     int count = ((Integer)tags.get(tag)).intValue();
   System.out.println("标签<" + tag + ">出现了" + count + "次");
  }
 }
 public static void main(String[] args) {
  try {
    XMLReader xmlReader= XMLReaderFactory.createXMLReader();
    xmlReader.setContentHandler(new SAXSample()); //注册事件处理程序
    xmlReader.parse("book.xml");  //解析 XML 文件
  } catch (Exception e) {
          e.printStackTrace();    }
 }
}
```

编译并运行上面的代码，得到结果如图 13-4 所示。

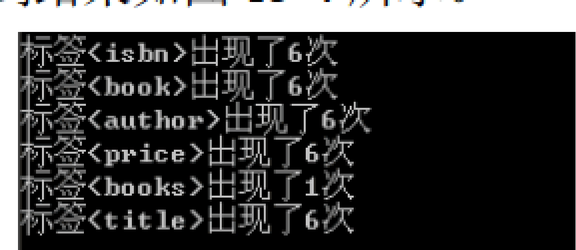


图 13-4　SAX 示例



13.5　上机练习

本章上机实验主要练习如何在 JSP 中使用 XML。其中应重点掌握如何定义 DTD 和 XML Schema、如何编写规范的 XML 文档以及如何使用 DOM 和 SAX 来操作 XML 文档等。

下面以使用 SAX 解析 XML 文件为例进行练习。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

(2) 在打开的 Eclipse 主窗口中，选择【File】|【New】|【Project】命令新建一个项目。

(3) 在打开的 New Project 对话框中，选择【Java】|【Tomcat Project】选项，单击 Next 按钮打开 New Tomcat Project 对话框。

(4) 在 New Tomcat Project 对话框的 Project Name 文本框中输入项目名称，如 testTomcat，单击【Finish】按钮，Eclipse 将自动创建一个 Tomcat 项目。

(5) 在 Eclipse 主窗口左侧的 Package Explorer 窗口中将出现新建的 testTomcat 项目。

(6) 选择【File】|【New】|【Other】命令，弹出 New 对话框创建 XML 文件。

(7) 在 New 对话框的 Wizards 列表中展开 XML 选项，选定 XML，单击【Next】按钮打开 Create XML File 对话框。

(8) 在 Create XML File 对话框中选中【Create XML file from scratch】复选框，单击【Next】按钮打开 Create XML File 对话框的 XML File Name 步骤。

(9) 在 File name 文本框中输入 XML 文件名，如 book.xml，单击【Finish】按钮创建 XML 文件。

(10) Eclipse 将创建一个空 XML 文件，仅包括 XML 文件头说明。在编辑窗口中输入 book.xml 的文档内容。

(11) 选择【File】|【New】|【JSP】命令，弹出 New JavaServer Page 对话框以创建 JSP 文件。

(12) 在 New JavaServer Page 对话框的 File name 文本框中输入文件名，如 SAXSample.jsp。

(13 单击【Finish】按钮新建一个 JSP 页面文件。

(14) Eclipse 会创建一个新的 JSP 文件，包括了基本的页面框架。在编辑窗口中根据 13.4 节中的例子编写 SAXSample.jsp 的页面代码(处理部分可直接调用 SAXSample 类)。

(15) 如果输入错误，Eclipse 会在发生错误的位置以红色下划波浪线表示，同时在错误行用红色显示。将鼠标移动到错误位置，会显示错误信息。

(16) 根据错误提示，修改所有可能的语法错误。

(17) 在 Eclipse 主窗口中选择【Tomcat】|【Start Tomcat】命令，启动 Tomcat 服务器。

(18) 打开 Internet Explorer 浏览器，输入相应的 URL，如 http://localhost:8080/testTomcat/SAXSample.jsp。观察页面运行效果。



13.6 习题

13.6.1 填空题

1. XML 中能出现的有效对象包括：________、________、________、________、________和________。

2. 一般来说，一个 XML 文档包括 3 个要素________、________和________。

13.6.2 选择题



1. 以下对于 XML 特性的描述，只有(　　　)是不正确的。

A. XML 是纯文本格式，与平台无关

B. XML 是可扩展的、自定义的文档，方便不同系统定义不同的标准文档

C. XML 将文档的数据、结构和显示方式集合在一起

D. XML 把数据存储方式不受显示格式的制约

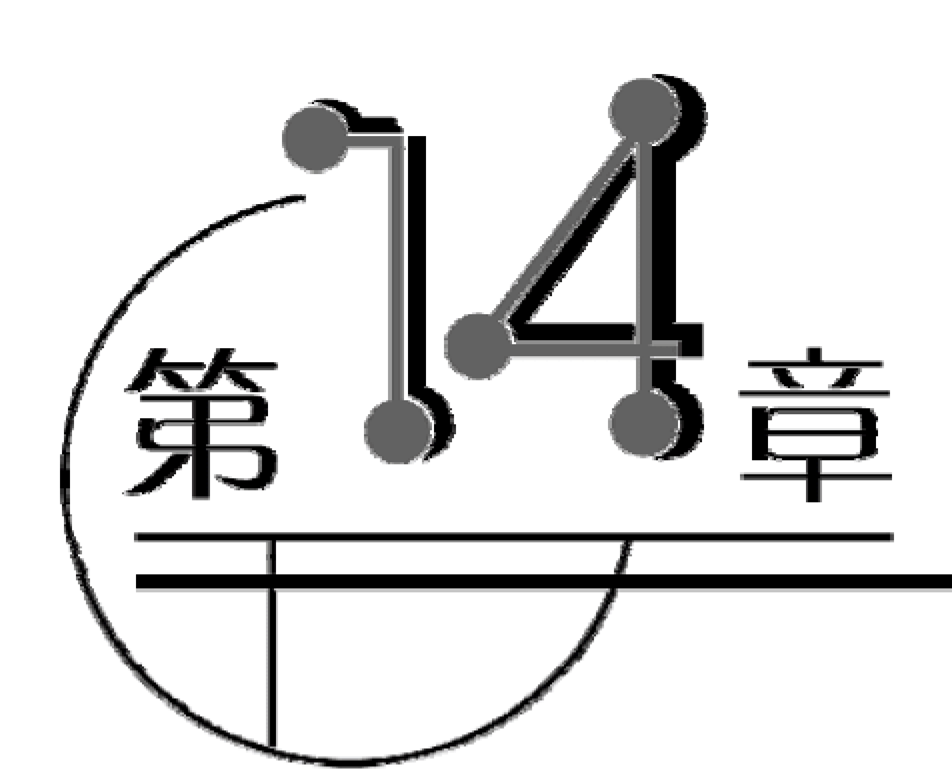
2. 以下哪些类是 DOM 对象(　　)(多选)。

A. Element　　B. Node　　C. NodeList　　D. DoCument

13.6.3 问答题

1. 简单比较 XML 和 HTML 的特性。

2. Schema 与 DTD 相比有什么优势？



JSP 应用的部署和错误处理

学习目标

JSP 灵活简便的 XML 配置文件机制为开发人员提供了对 JSP 应用程序运行方式的控制权和方便部署，开发人员将设置置于配置文件中，以后更改时无需重新编译应用程序。另外，JSP 页面执行时可能会碰到并非程序本身引起的错误，如后台数据库发生问题无法访问，这就需要一种有效的机制及时捕获并处理这些错误。Java 提供非常强大的异常处理机制来解决这个问题，使开发人员能够通过简单的几个语句就能完成所有的后台处理工作。本章将详细讲解 JSP 的配置方法和过程，JSP 的调试方法和技巧以及 JSP 的异常处理机制等。

本章重点

- JSP 应用的部署
- JSP Web 应用配置
- JSP 错误处理

14.1 JSP 高级配置和部署

本节将介绍 JSP 的高级配置和程序的部署。

14.1.1 JSP Web 应用程序综述

Web 应用程序是指使用独立的 URL 前缀(很多商业网站都使用域名后加虚拟目录名的方式

来区分网站上不同的应用，例如 www.mysite.com/petstore 和 www.mysite.com/forum)，拥有独立的配置文件和标准目录组织形式，通常在 Web 服务器的单独内存空间运行的应用。

Web 应用程序需要运行在 Web 服务器上，因此 JSP Web 应用程序需要支持 JSP 引擎的 Web 服务器。市场上有众多支持 JSP 的 J2EE 应用服务器和 Web 服务器，如 BEA 的 WebLogic 和 IBM 的 WebSphere 等都是应用非常广泛的 J2EE 应用服务器，一些第三方供应商也提供了用于微软 IIS 服务器的 JSP 引擎。这些服务器对于 JSP Web 应用程序具有不同的配置管理方式，下面以使用最为广泛的开源软件 Tomcat 为例来介绍如何部署和配置 JSP 应用程序。

原则上，JSP 开发人员可以按照自己的方式任意组织 Web 应用程序。但为了使程序的部署和配置更方便快捷，减少书写大量的控制逻辑以节省开发时间，更简单地在不同的应用服务器之间迁移，就需要一种统一的应用程序结构定义。J2EE 规范在发展的过程中也充分考虑到了这一点，为此定义了目录结构的标准：



- 所有文件都位于 document root(文档根)目录下，可以用 J2EE SDK 提供的工具将整个应用程序打包为一个.war 文件。war 是 Web Application aRchive 的缩写，表示 Web 应用程序的一种压缩文件格式。
- *.jsp, *.html, etc.：静态和动态(主要是 JSP)页面文件以及其他所有对于客户浏览器可见的文件(包括图片，脚本文件 Javascript，样式表文件等)都放置在根目录下。对于较大的应用程序，可以在根目录下建立更为复杂的目录层次结构。对于简单的应用程序则完全可以简单地将这些文件放在根目录下。
- /WEB-INF/web.xml：web.xml 文件用于配置 Web 应用程序，它被称为 Web 应用程序部署描述器(Web Application Deployment Description)。它是一个用来描述 Servlet 和其他 Web 应用程序组成部分以及它们的初始参数等属性的 XML 文档。在下面的章节中将会详细介绍该文件的主要内容结构、如何配置与访问应用配置项。
- /WEB-INF/classes：这个子目录用于存储所有的 Java 类文件和相关资源文件，如图片、语言信息等。这些类文件可能是 Servlet，也可能是普通的 Java 类。需要注意的是，如果一个类文件属于某个包(package)，则需要将整个目录层次结构放置于 classes 目录下。如 com.mycompany.mypackage.myclass 类需要对应/WEB-INF/classes/com/mycompany/mypackage/myclass.class 文件。
- /WEB-INF/lib：该子目录用于存放 Web 应用程序所需的所有库文件，这些库文件是以压缩的.jar 文件格式储存的，它包含所有 Web 应用程序必需的类文件和相应的资源文件。例如，一个电子商务应用需要访问 Oracle，就需要将要使用的 JDBC 驱动程序库文件都放置于 lib 目录下。

14.1.2 JSP Web 应用部署

下面介绍如何将 JSP Web 应用程序部署到 Tomcat 5Web 服务器上。部署的方法包括以下几种：



- 直接将开发机器上该应用的整个目录层次结构拷贝到部署该应用的服务器的 Tomcat 安装目录 webapps 目录下的一个子目录，如 C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\MyWebApp。Tomcat 会基于所选择子目录名字为 Web 应用程序自动分配一个上下文路径(虚拟路径)，如上述的目录对应于以下的 URL: http://localhost:8080/MyWebApp。

提示

使用这种方式部署时，要确保相应的库文件位于 WEB-INF/lib 目录下。部署之后需要重新启动 Tomcat 服务。

- 使用 J2EE SDK 所带的 package 工具或者 Eclipse 直接生成 Web 应用存档文件(.war 文件)。将.war 文件直接拷贝到 Tomcat 安装目录的 webapps 目录下。一旦启动 Tomcat，它会自动创建一个与.war 文件同名的目录，解压.war 文件内容到该目录，并执行该应用。



提示

使用这种方式部署时，如果需要更新应用程序，要确保先删除该目录下所有内容并替换.war 文件，重新启动 Tomcat 则自动对应用进行更新。

下面简要介绍如何使用 Eclipse 来创建.war 文件。

(1) 假设已经创建了一个名为 testTomcat 的 Web 应用，并在本机调试完成。选择【Project】|【Properties】命令，如图 14-1 所示。

(2) 在打开的 Properties for testTomcat 对话框中选择【Tomcat】选项，如图 14-2 所示。

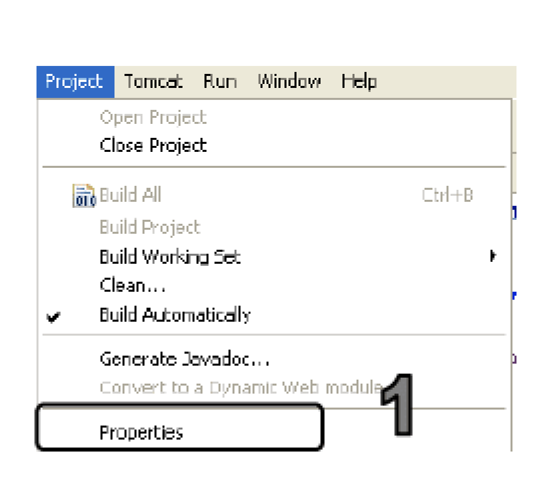


图 14-1　Project 菜单

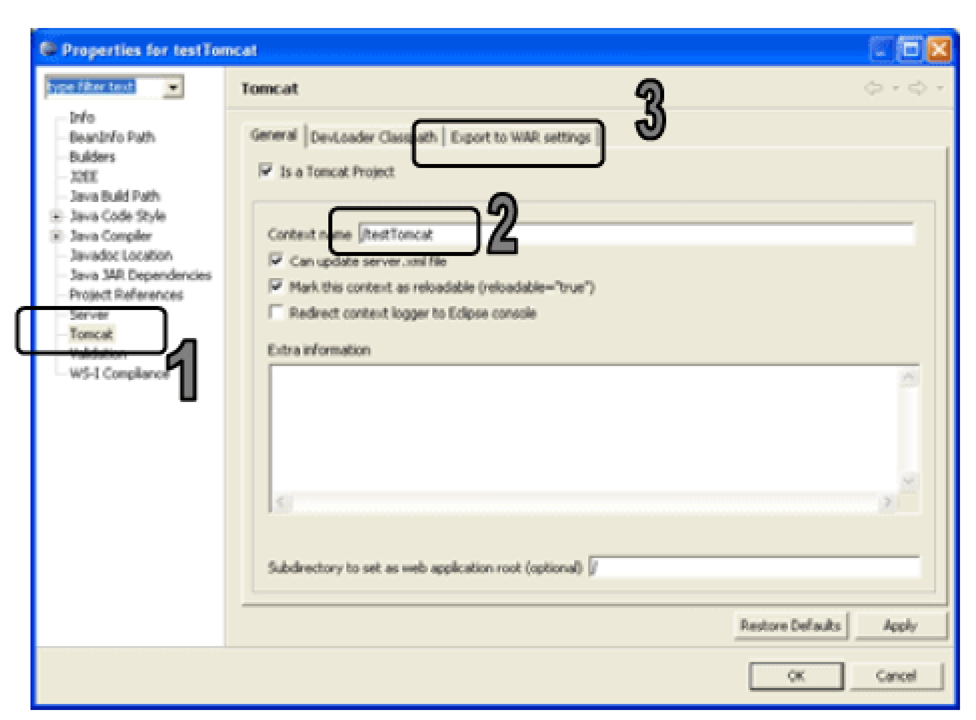


图 14-2　项目属性对话框-Tomcat-General 选项

(3) 确保内容如图 14-2 所示，或修改其 Context name 为部署时希望使用的名字。单击【Export to WAR settings】选项卡，如图 14-3 所示。

(4) 输入或单击【Browse】按钮选择导出至 WAR 文件的目标文件名，单击【OK】按钮完成项目属性配置。在 Package Explorer 窗口中右键单击 testTomcat 项目名，选择【Tomcat Project】

|【Export to the WAR file sets in project properties】命令即可导出.war 文件，如图 14-4 所示。

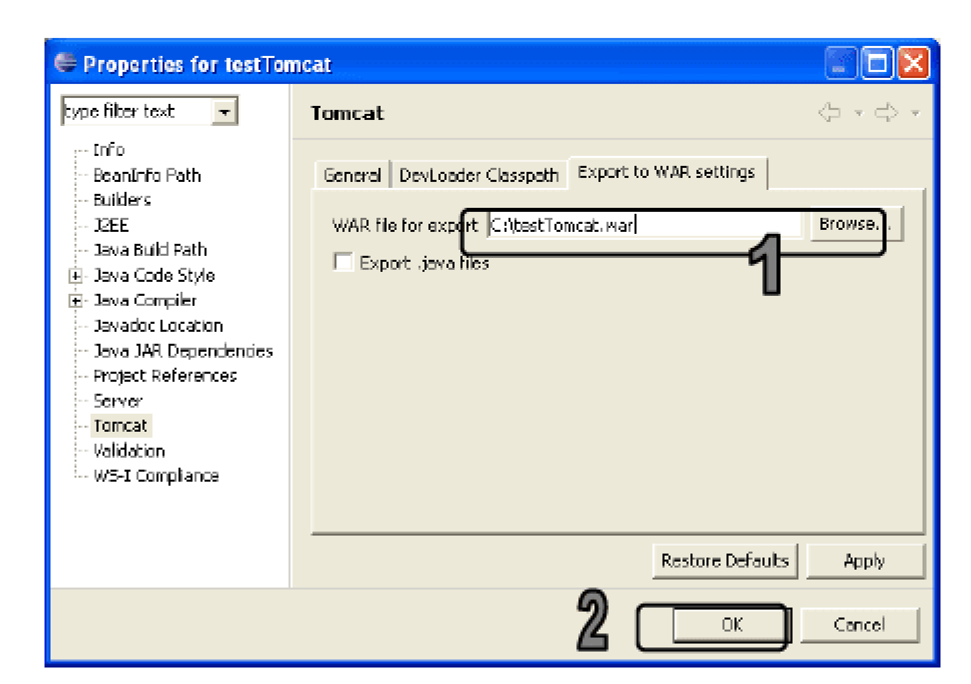


图 14-3　项目属性对话框-Tomcat-Export to WAR Settings 选项

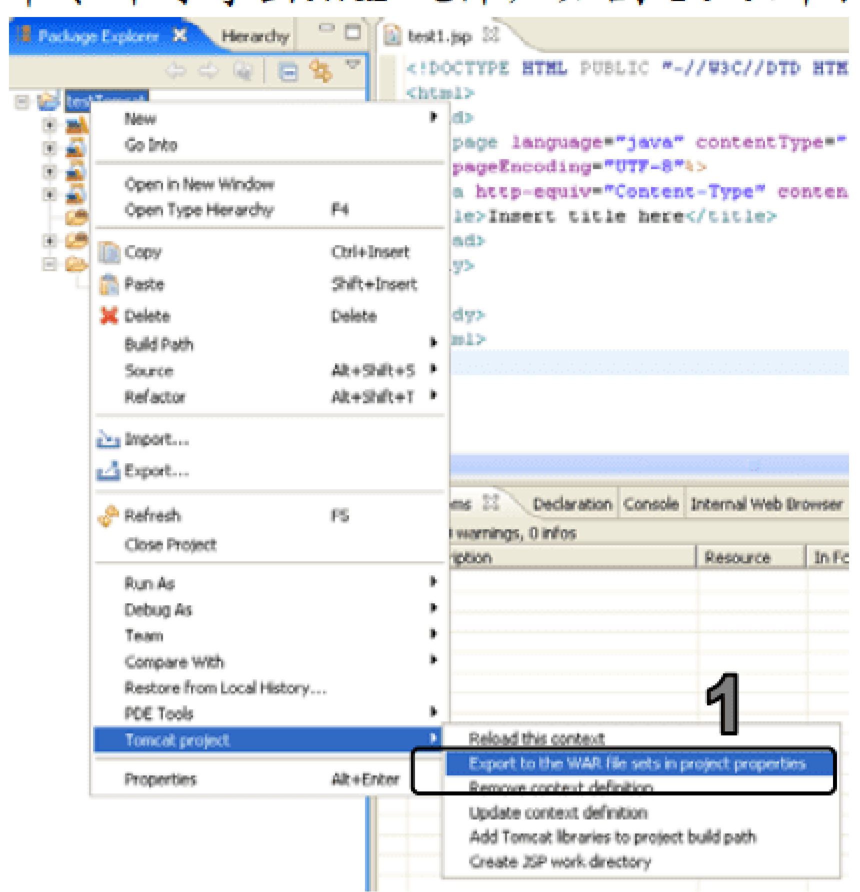


图 14-4　导出 war 文件菜单命令

(5) Eclipse 成功导出.war 文件后，将弹出如图 14-4 所示的对话框，单击【OK】按钮。找到步骤(3)中定义的 war 文件，即可将其拷贝部署到运行环境下。

◉ Tomcat 应用服务器还提供了一个 manager 应用程序，用来管理 Tomcat 上的应用程序。也可以通过它来部署 Web 应用。当然这里的部署还是需要.war 文件，与上面的方法不同之处在于使用 manager 应用进行部署时可以不用重新启动 Tomcat 服务。在实际运行的网站上更新原有应用或部署新的应用的情况下这是一种非常方便且有效的方法，因为一些实际的商业应用一旦重启服务，带来的损失是难以估量的。下面就简单介绍一下如何通过 Tomcat Manager 应用程序部署 Web 应用。

(1) 使用 IE 浏览器访问已安装的本机 Tomcat 服务器，默认的 URL 为：http://localhost:8080。在打开的 Tomcat 默认页面的左上方，有一个 Administration 区域，要用到的 Tomcat Manager 应用程序的链接就在这块区域中，如图 14-6 所示。

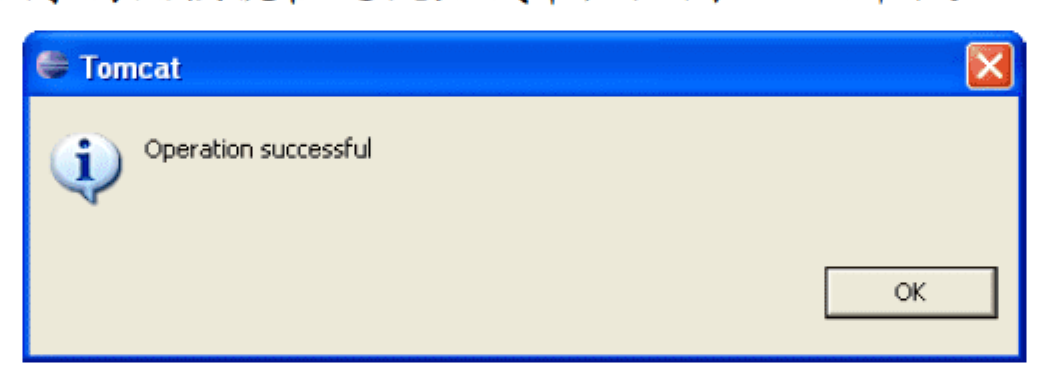


图 14-5　操作成功对话框



图 14-6　Tomcat Manager 链接

(2) 单击 Tomcat Manager，将出现如图 14-7 所示的 Connect to localhost 对话框。使用 Tomcat 默认的管理员用户 admin，密码为空即可登录。

(3) 登录之后会显示如图 14-8 所示的 Manager 应用程序界面。默认的页面就是 List Applications 页面，列出了当前可用的所有 Web 应用。



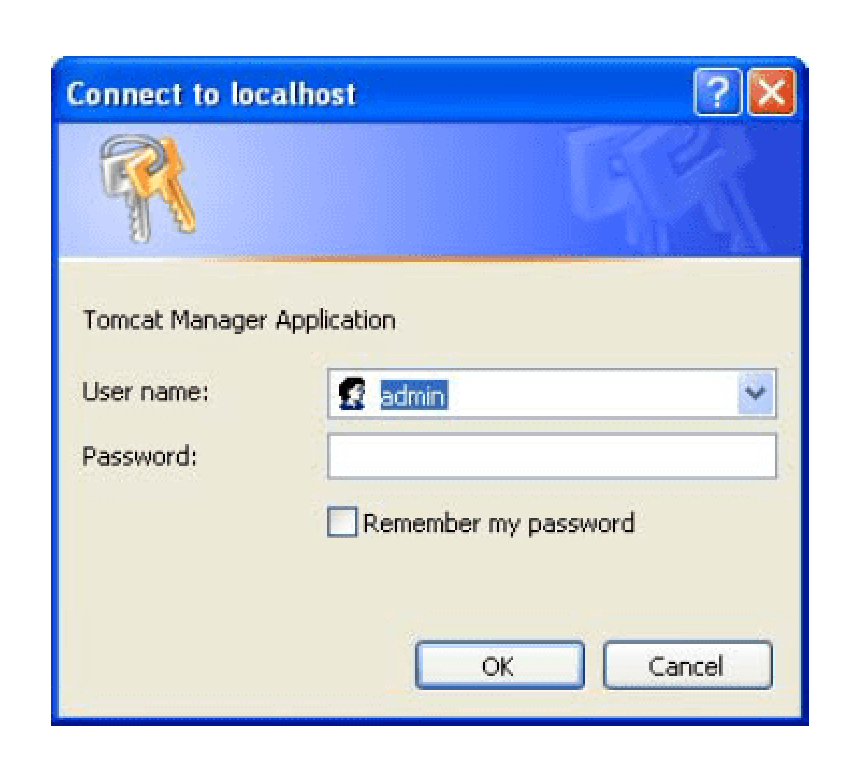


图 14-7　Connect to localhost 对话框

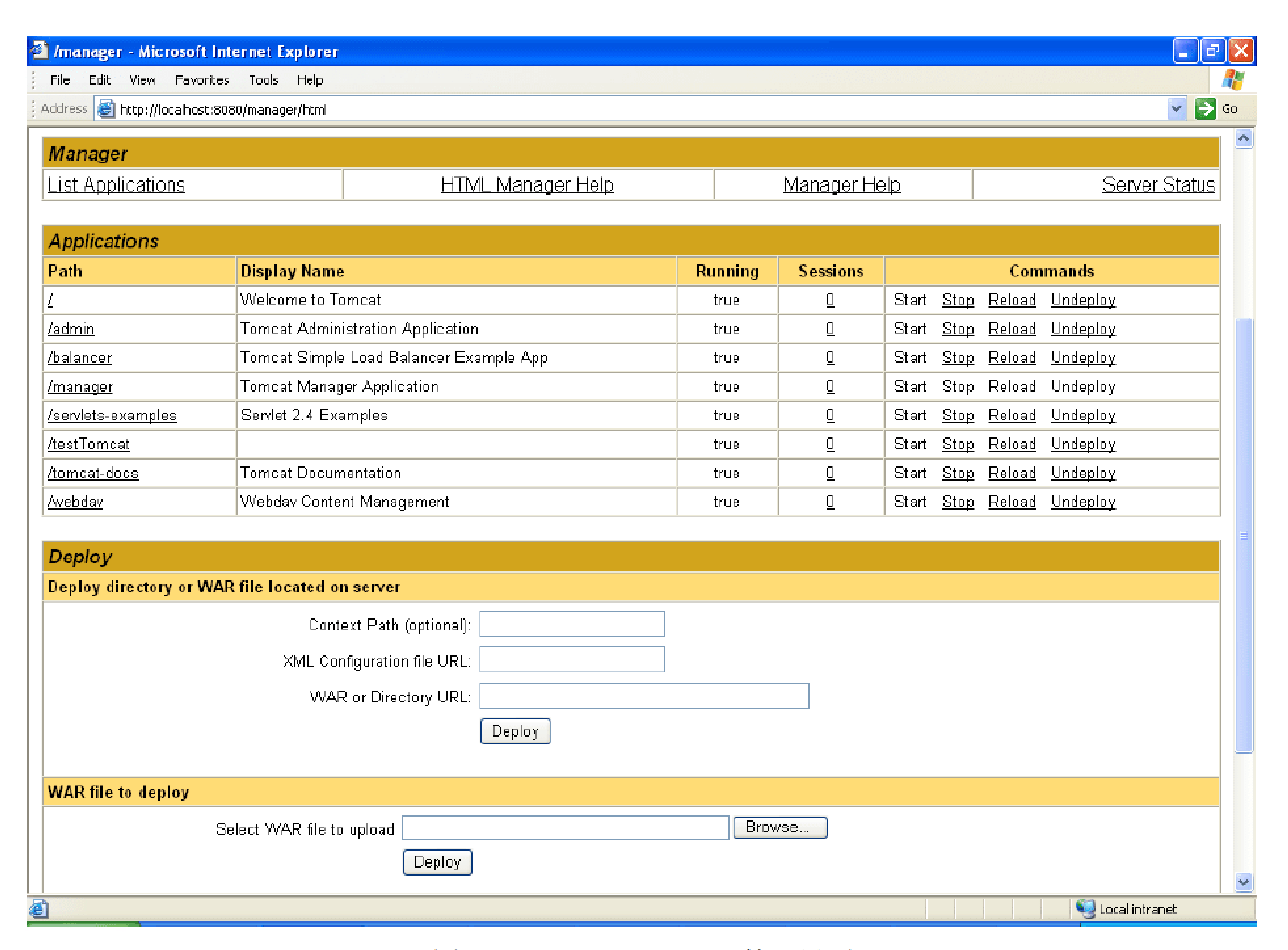


图 14-8　Tomcat Manager 管理界面



(4) 从图 14-8 中可以看到在 Applications 区域列出了当前 Tomcat 服务器所有的 Web 应用程序。Path 列表示其 URL 的虚拟路径，Display Name 显示了 Web 应用的名字，Running 表示其运行与否的状态，Sessions 表示当前应用进程的个数，Commands 当中包括了 Start(启动)、Stop(停止)、Reload(重载)或 Undeploy(卸载)Web 应用的命令。

(5) 在 Deploy 区域可以对目录和 WAR 文件进行部署。在 WAR file to deploy 区域的 Select WAR file to upload 文本框中，单击【Browse】(浏览)按钮选择需要部署的 Web 应用程序文件，



如 d:\test.war 文件。单击【Deploy】(部署)按钮，Tomcat 服务器就会自动将 WAR 文件所包含的 Web 应用程序部署到相应的路径下。

14.1.3 JSP Web 应用配置

对于任何 Web 应用程序，服务器的主要要求是具有丰富而灵活的配置系统——使开发人员能够轻松地将设置与可安装的应用程序关联起来并使管理员能够在部署后轻松地自定义这些值。JSP Web 应用系统提供一个分层配置结构以满足相关人员的需要，使得开发人员能够在整个应用程序、站点或计算机中定义和使用可扩展的配置数据。JSP 的配置系统提供了以下好处：

- 配置信息存储在基于 XML 的文本文件之中，可以使用任何标准的文本编辑器或 XML 分析器来创建和编辑 JSP 配置文件。
- 在运行时，JSP 使用虚拟目录结构 WEB-INF 子目录下的 web.xml 文件提供的配置信息为每个惟一的 URL 资源分配并缓存结果配置设置，以供所有后续对资源的请求使用。
- JSP 配置系统是可以扩展的，可以定义新的配置参数对它们进行处理。
- JSP Web 应用中 WEB-INF 子目录下的文件禁止直接通过浏览器访问，从而保护了配置文件不受外部访问。

JSP 的配置系统包括站点级和应用级两个层次的配置文件。所有的配置文件都使用 XML 格式。站点级的配置文件放置于 Tomcat 安装路径的 conf 子目录下，包括 context.xml，jkconf.ant.xml，server.xml，server-minimal.xml，tomcat-users.xml 和 web.xml。通常可能需要修改配置的主要是 server.xml 与 web.xml 两个文件，下面就对他们进行详细介绍。

- conf/server.xml

下面的代码是从 Tomcat 服务器的 server.xml 中抽取的部分常用配置项的值：

```
//server.xml
<!--服务器 Server-->
<Server port="8005" shutdown="SHUTDOWN">
  <!--服务 Service-->
  <Service name="Catalina">
    <!--连接器 Connector-->
    <Connector
      port="8080"
      maxHttpHeaderSize="8192"
      maxThreads="150"
      minSpareThreads="25"
      maxSpareThreads="75"
      enableLookups="false"
```






```
        redirectPort="8443"
        acceptCount="100"
        connectionTimeout="20000"
        disableUploadTimeout="true" />
      <!--引擎 Engine -->
      <Engine
        name="Catalina"
        defaultHost="localhost">
        <!--主机 Host -->
        <Host
          name="localhost"
          appBase="webapps"
          unpackWARs="true"
          autoDeploy="true"
          xmlValidation="false"
          xmlNamespaceAware="false">
          <!--上下文 Context -->
          <Context
            path="/testTomcat"
            reloadable="true"
            docBase="F:\Work\workspace\testTomcat"
            workDir="F:\Work\workspace\testTomcat\work" >
          </Context>
        </Host>
      </Engine>
    </Service>
  </Server>
```



Server 配置项代表整个 Tomcat 服务器，它在一个指定的端口(如本例中的 8005)监听 SHUTDOWN 命令。

Service 是一个或多个 Connector(连接器)的集合，它代表一个 Tomcat 的 Web 服务。默认情况下，Tomcat 服务器安装时会使用 Catalina 作为其服务名。

Connector(连接器)是 Web 服务器接收 HTTP 请求以及发送 HTTP 响应的一个端点。默认情况下，Tomcat 会在 8080 端口建立一个 HTTP 1.1 的连接器，所有客户端的请求都由这个连接器来响应。连接器的类型还包括 SSL HTTP 连接器(用于建立安全连接)、AJP 连接器和代理服务器的 HTTP 连接器等。

事实上连接器只是建立了与客户请求的连接，真正的处理(尤其是 JSP 和 Servlet 的运行)都是通过 Engine(引擎)来完成的。Engine 分析请求的头信息，并将其传送到相应的虚拟主机上处

理。Engine 的属性值包括 name(引擎名)和 defaultHost(默认主机名)。Engine 可能包括多个 Host (主机)子节点，它可能的子节点还包括 Valve 和 Realm 等。

Host 定义了虚拟主机的属性，它的主要属性值包括：name(主机名)、appBase(Web 应用的路径，默认为 webapps)、unpackWARs(是否解压 WAR 文件)、autoDeploy(是否自动部署)和 xmlValidation(部署时是否对 xml 配置文件进行校验)等。

Context(上下文)是 Host 的子节点，它定义了一个 JSP Web 应用程序的基本部署情况。一般而言，希望运行在非默认目录(webapps)下的 Web 应用才需要在 server.xml 中配置该项。Context 的属性值包括：path(Web 应用的虚拟路径)、reloadable(是否可重载，即如果替换 WEB-INF 下的配置文件，类和库文件等，不用重启 Tomcat 服务)、docBase(Web 应用程序文件所在的目录)和 workDir(Web 应用程序的工作目录)。

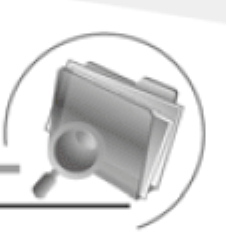
◉ conf/web.xml

前面已经提到 conf 目录下的配置文件是用于站点级的配置，web.xml 也不例外。所以，对 web.xml 进行配置时一定要非常小心，要确保是对该站点上所有 Web 应用都有效的配置项才在此配置。下面节选了一部分 conf/web.xml 的内容作简要介绍：

```
//conf/web.xml
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

<!-- Servlet 定义(JSP)-->
  <servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
      <param-name>fork</param-name>
      <param-value>false</param-value>
    </init-param>
    <init-param>
      <param-name>xpoweredBy</param-name>
      <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
  </servlet>

<!-- Servlet 映射(JSP 映射)-->
  <servlet-mapping>
    <servlet-name>jsp</servlet-name>
```



```
        <url-pattern>*.jsp</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>jsp</servlet-name>
        <url-pattern>*.jspx</url-pattern>
    </servlet-mapping>

<!-- 进程 session 配置-->
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

<!-- MIME 映射-->
    <mime-mapping>
        <extension>doc</extension>
        <mime-type>application/msword</mime-type>
    </mime-mapping>

<!-- 欢迎文件列表-->
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```



上述 conf/web.xml 文件中根节点是 web-app，表示这是一个 Web 应用的配置文件。下面将其内容分块进行介绍：

第一部分是 Servlet 的定义部分，主要是对 Servlet 的一些默认设置进行配置。前面已经介绍过，jsp 页面事实上是被 Tomcat 转变为 Servlet 来处理，所以对于 JSP 也有相应的 Servlet 定义配置，如上述 conf/web.xml 文件中所示。可以在此配置一些有用的初始参数值。

Servlet 定义之后，则需要将相应的 Servlet 程序映射到某个 URI 上，即第二部分。这里节选了关于 JSP 的映射。上述 conf/web.xml 文件中将*.jsp 和*.jspx 的文件都映射到了名为 jsp 的 Servlet 上，而这个 jsp Servlet 就是在第一部分中定义的。所以当 Tomcat 接收到*.jsp 或*.jspx 文件的请求时，就会调用 JspServlet 程序对文件进行解析并运行。

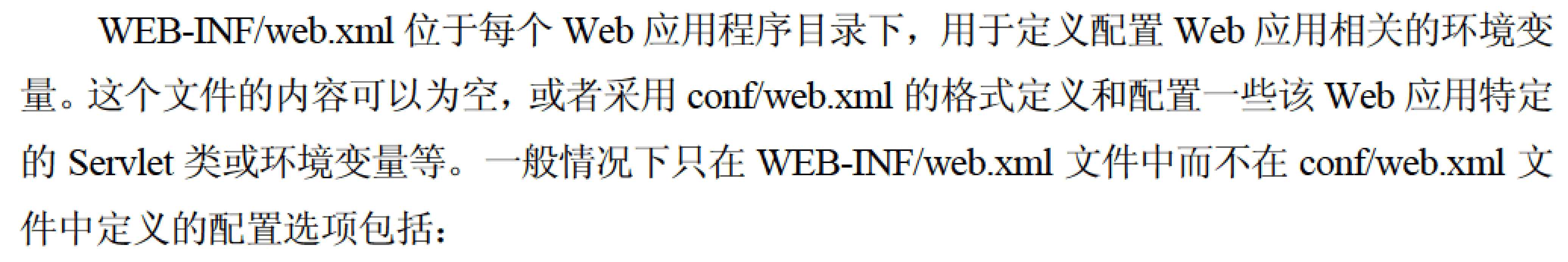
第三部分则是对进程 Session 的一些属性的设置，最常用的就是对 Session 的超时的设置。

第四部分定义了 MIME 类型的映射。相信现在很多读者都已经熟悉 MIME 这个词，它是 Multipurpose Internet Mail Extensions(多用途因特网邮件扩展)的缩写。最初 MIME 被用来开发在电子邮件中传递非文本信息，现在，所有的 Web 服务器和客户机都使用 MIME 来定义需要交

换的信息类型。一个 MIME 类型是一个 type/sub-type(类型/子类型)的字符串，如 text/html、image/jpg、application/word 等。客户端在接收到 MIME 数据时，根据其提供的文件类型调用相应的应用程序来处理。在这里 MIME 映射的意思就是通过将文件扩展名和 MIME 类型映射起来，客户端可以根据响应发送的文件名后缀来判断信息类型。

最后一部分定义了目录下的欢迎文件列表。多数网站都提供了使用简单的 URL 和路径名来访问站点的方法，例如访问 Java 官方网页只需输入 http://java.sun.com 即可，而不用输入如 http://java.sun.com/index.jsp。实际上，访问该网址时服务器端会自动寻找欢迎文件并请求之，通常的欢迎文件会以 index.htm、index.jsp、default.jspx、toc.html 等方式出现，也可以定义任何文件为欢迎文件，如 Iamnotwelcomepage.htm。

◉ WEB-INF/web.xml



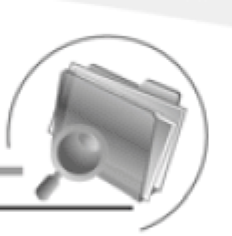
WEB-INF/web.xml 位于每个 Web 应用程序目录下，用于定义配置 Web 应用相关的环境变量。这个文件的内容可以为空，或者采用 conf/web.xml 的格式定义和配置一些该 Web 应用特定的 Servlet 类或环境变量等。一般情况下只在 WEB-INF/web.xml 文件中而不在 conf/web.xml 文件中定义的配置选项包括：

(1) 上下文初始化参数。上下文参数是 Servlet 的部分属性，在该 Web 应用程序中所有的 Servlet 和 JSP 页面都可以简单方便地获取和设置这些值。通过<context-param>元素可以定义上下文参数。如下面的代码定义了一个值为 100 的 param1 属性：

```
<web-app>
...
  <context-param>
    <param-name> param1</param-name>
    <param-value>100</param-value>
    <description>param1 参数，仅用于测试</description>
  </context-param>
...
</web-app>
```

这些上下文参数可以在 Servlet 或者 JSP 页面中，通过 getInitParameters()和 getInitParameter (name)方法来读取。<param-name>是上下文参数的名称，即 getInitParameter()方法中用来检索参数使用的名字。<param-value>则是上下文参数的值，即 getInitParameter 的返回值。<description>是对参数的描述，使其更具可读性。这些上下文参数被广泛应用于存储全局常变量值，比如 JDBC 驱动程序配置项，简单 Web 应用的用户信息等。

(2) 错误页面。错误页面是指可以在部署描述器中配置遇到相应错误时转向的错误处理页面。虽然错误也可以在 Servlet 和 JSP 页面中处理，不过缺少统一的管理容易使页面的错误处理混乱，另外在每个页面中需要对每种可能的错误进行编程处理也会带来很多不必要的工作量和错误。JSP Web 应用程序可以利用<error-page>元素的配置来解决这个问题。



对于每一个 HTTP 错误或者异常，都可以指定一个错误页面(静态或动态，甚至是图片或其他)的 URL。当 Tomcat 服务器监听到相应的错误或异常时，就会把相应的资源通过 HTTP 响应(Response)发送给客户。<error-page>可以通过检测异常类型或者 HTTP 错误代码来定义错误页面，如下所示：

```
<web-app>
...
  <error-page>
    <exception-type>javax.servlet.ServletException</exception-type>
    <location>/errors/servletException.htm</location>
  </error-page>
  <error-page>
    <error-code>500</error-code>
    <location>/errors/internalError.htm</location>
  </error-page>
...
</web-app>
```





14.2　JSP 错误处理

当页面出现错误时，JSP 会将与该错误有关的信息发送到客户端，同时自动在后台的日志文件中记录错误详细信息。错误分为如下 3 类。

配置错误：当 Web 应用程序的配置文件 web.xml 的语法或结构不正确时发生。

编译错误：当页面目标语言中的语句语法不正确时发生。

运行时错误：在页的执行过程中发生，在编译时无法检测到该错误。

14.2.1　配置错误

配置错误是指在书写 web.xml 文件时引入一些错误，比如一个标签没有书写对应的结束标签，或者因为笔误导致结束标签与开始标签不一致等。这种情况下运行 JSP Web 应用程序必然要出现错误。例如，尝试在一个 Web 应用程序的 WEB-INF/web.xml 文件中加入一个<test>标签，而没有结束标签。然后重新启动该应用程序，访问应用中的任意页面时，结果将出现如图 14-9 所示的页面。

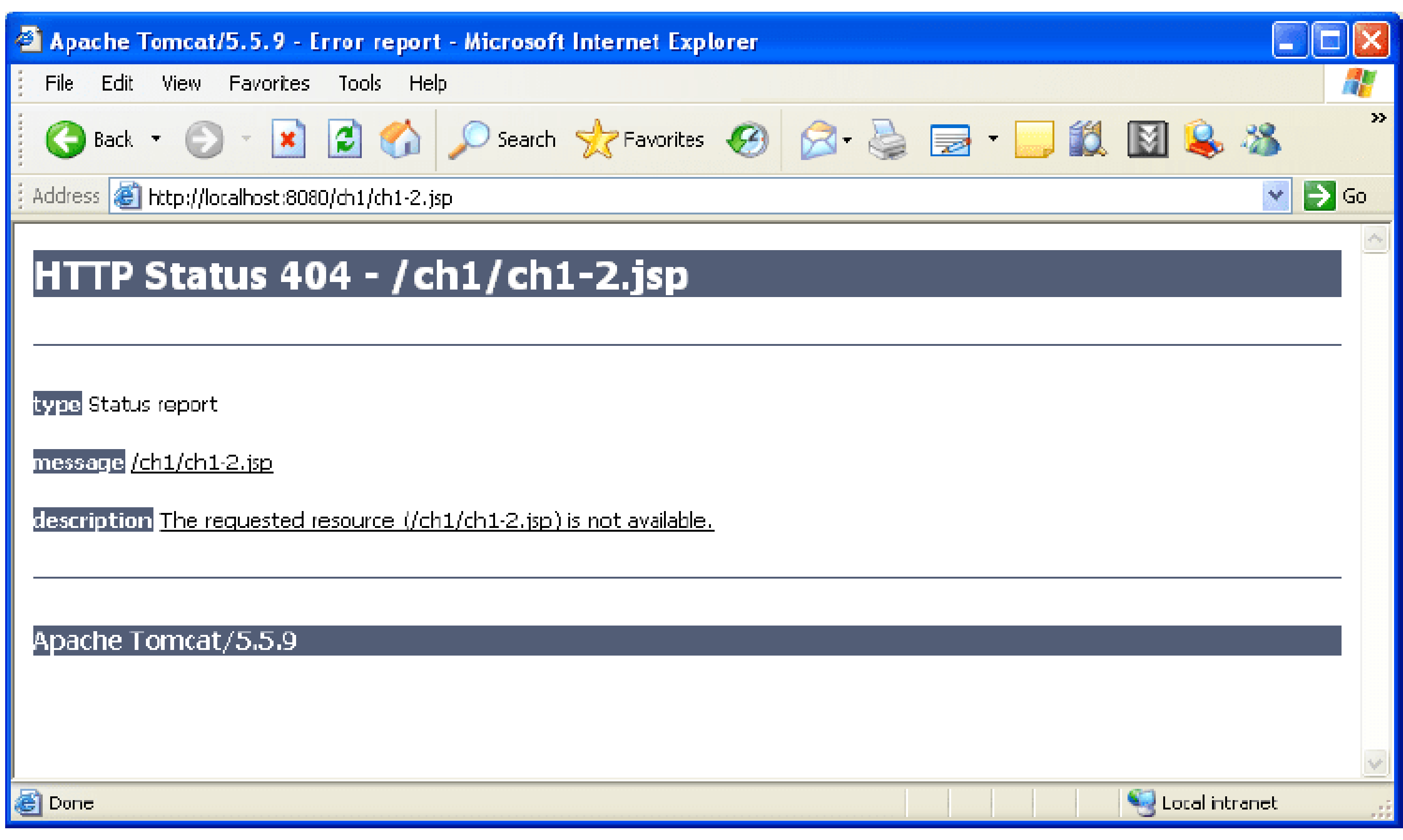


图 14-9　配置错误时显示的页面



在 Tomcat 安装路径下的 logs 子目录中可以找到所有的日志信息，打开当前使用的标准错误日志文件(格式为 stderr_YYYYMMDD.log)。可以找到以下的日志记录：

```
2006-1-1 22:13:13 org.apache.catalina.startup.ContextConfig applicationWebConfig
严重: Parse error in application web.xml
org.xml.sax.SAXParseException: The element type "test" must be terminated by the matching end-tag "</test>".
```

这些信息提示在 web.xml 中<test>元素必须要有一个对应的结束标签</test>。如图 14-10 所示，显示了部分错误提示信息的情况。

```
2006-1-1 22:13:13 org.apache.tomcat.util.digester.Digester fatalError
严重: Parse Fatal Error at line 45 column 3: The element type "test" must be terminated by the matching end-tag "</test>".
org.xml.sax.SAXParseException: The element type "test" must be terminated by the matching end-tag "</test>".
        at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.createSAXParseException(Unknown Source)
        at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.fatalError(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLErrorReporter.reportError(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLErrorReporter.reportError(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLScanner.reportFatalError(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanEndElement(Unknown Source)
        at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl$FragmentContentDispatcher.dispatch(Unknown
        at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScannerImpl.scanDocument(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(Unknown Source)
        at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.parse(Unknown Source)
        at org.apache.tomcat.util.digester.Digester.parse(Digester.java:1561)
        at org.apache.catalina.startup.ContextConfig.applicationWebConfig(ContextConfig.java:339)
        at org.apache.catalina.startup.ContextConfig.start(ContextConfig.java:1031)
        at org.apache.catalina.startup.ContextConfig.lifecycleEvent(ContextConfig.java:255)
        at org.apache.catalina.util.LifecycleSupport.fireLifecycleEvent(LifecycleSupport.java:119)
        at org.apache.catalina.core.StandardContext.start(StandardContext.java:4053)
        at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1012)
        at org.apache.catalina.core.StandardHost.start(StandardHost.java:718)
        at org.apache.catalina.core.ContainerBase.start(ContainerBase.java:1012)
        at org.apache.catalina.core.StandardEngine.start(StandardEngine.java:442)
        at org.apache.catalina.core.StandardService.start(StandardService.java:450)
        at org.apache.catalina.core.StandardServer.start(StandardServer.java:683)
        at org.apache.catalina.startup.Catalina.start(Catalina.java:537)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:271)
        at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:409)
2006-1-1 22:13:13 org.apache.catalina.startup.ContextConfig applicationWebConfig
严重: Parse error in application web.xml
org.xml.sax.SAXParseException: The element type "test" must be terminated by the matching end-tag "</test>".
        at com.sun.org.apache.xerces.internal.parsers.AbstractSAXParser.parse(Unknown Source)
        at org.apache.tomcat.util.digester.Digester.parse(Digester.java:1561)
        at org.apache.catalina.startup.ContextConfig.applicationWebConfig(ContextConfig.java:339)
        at org.apache.catalina.startup.ContextConfig.start(ContextConfig.java:1031)
        at org.apache.catalina.startup.ContextConfig.lifecycleEvent(ContextConfig.java:255)
```

图 14-10　配置错误时的错误日志



根据这个错误提示修改 web.xml 配置文件后重新启动 Tomcat，再次访问该页面即可看到正确的页面显示。事实上，如果采用一个比较好的 XML 编辑器去编辑配置文件就可以避免绝大多数的配置错误。

14.2.2 编译错误

编译错误是由于在书写 JSP 文件或 Servlet、Bean 等 Java 程序时，使用了错误的语法而造成的。编译错误是最容易碰到的一种错误，因为在书写比较复杂的程序时很难保证自始至终语句的语法都正确，很多编译错误都来自一个笔误。

举一个简单的例子来说明一下。下面的一行代码来自于本书前面的一个 jsp 实例，但是漏打了一个@符号，所以访问该页面时，将出现如图 14-11 所示的错误信息。

```
<% page import="java.io.OutputStream" %>
```



这个错误页面给出了导致错误发生的原因，根据其描述的文件及位置可以找出错误发生的根本原因。也可以访问 logs 目录下的 localhost_YYYYMMDD.log 日志文件，在该文件中记录了错误的更为详细的信息。当然，由于很多情况下编译错误发生在将 JSP 页面转换为 Servlet 程序之后和将其编译为 class 文件期间，所以报出的错误从字面上也许会比较难以理解。这种情况下，就需要多对 JSP 源程序和转化后的 Servlet 程序(位于 work 目录相应的子目录下)进行认真细致的分析以找出真正的原因所在。

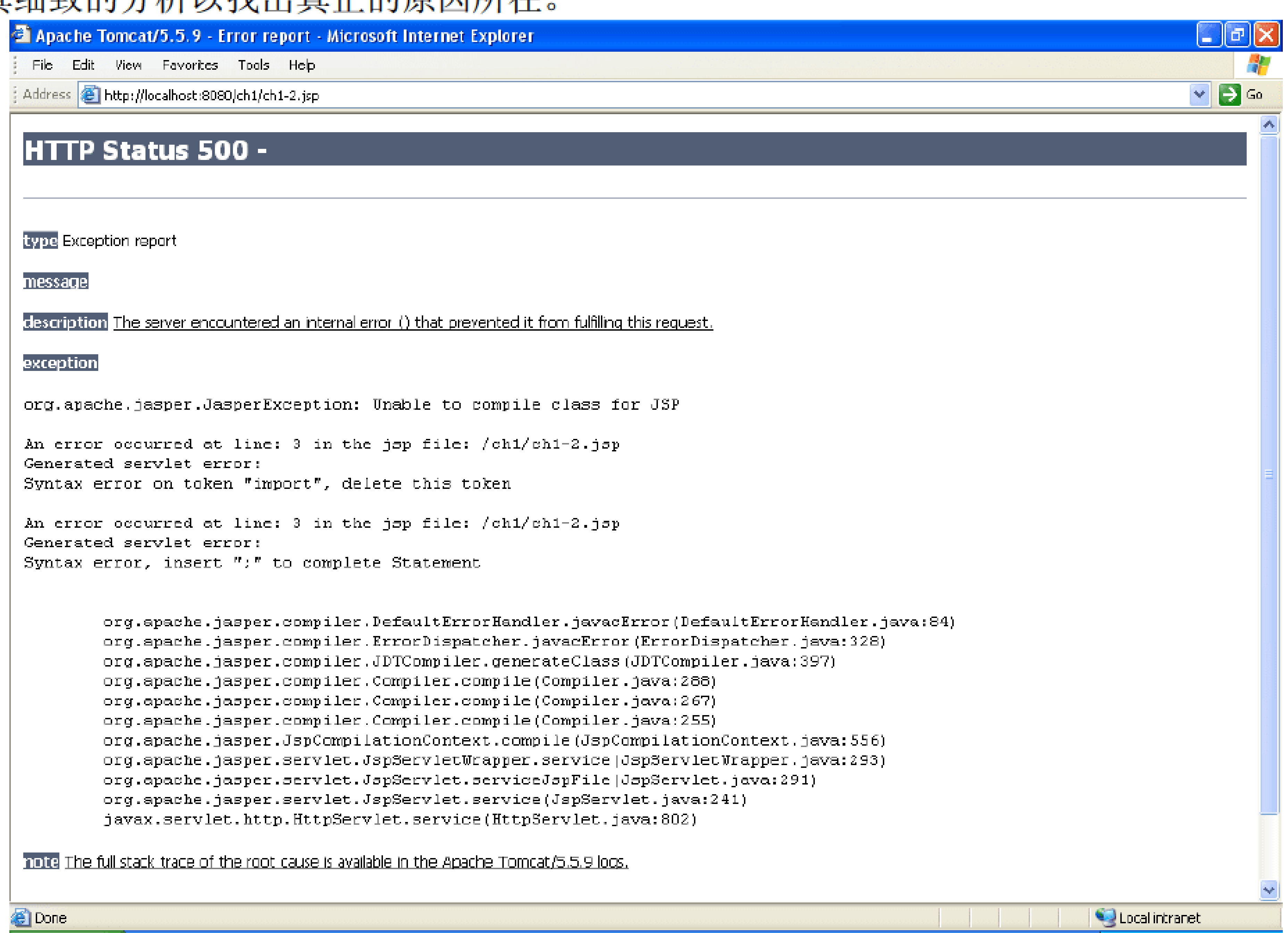


图 14-11 编译错误

14.2.3 运行时错误

对于运行时错误，有些情况下是由于程序的逻辑错误造成的，能够经过一定时间的调试找出原因并进行纠正，但在部署应用程序之前不一定能发现所有的逻辑错误。还有一些运行时错误是由其他的软硬件故障或外部不可控因素造成的，无法通过调试来保证它的正常运行。这就需要对这些错误的发生做一定的处理，以确保应用程序还能继续正常运行或提供给用户系统无法运行的正确信息。

发生运行时错误时可能看到错误页面提示，但是程序运行时 Tomcat 服务器是使用生成的 Servlet 程序来响应客户请求，所以，所有的错误提示都是和 Servlet 相关的。而生成的 Servlet 程序即使方便找到也是难以阅读的，尤其是 JSP 开发人员可能并不擅长 Java 程序的开发与调试。这种情况下就需要使用一些程序调试的方法和技巧。

14.2.4 JSP 调试方法和技巧简介

说起调试，很多人最先想到的就是集成开发调试程序，似乎今天的集成调试工具已经无所不能。但是 JSP 的特殊运行方式决定了它的调试也具有一定的特殊性，虽然已经有很多集成开发工具在 JSP 调试方面进行了深入的尝试，但调试一个服务器端应用程序永远不会很容易。另一方面，即使再完美的工具也不能保证程序编写质量的提高，相反地，工具可能给开发人员带来一定的惰性。而传统的调试方法和技巧却能给开发人员带来开发水平的提高从而减少错误的发生，也不失为一种实用的选择。下面简单介绍一些可用于 JSP 调试的传统方法和技巧。

◉ 代码审核

虽然看起来简单，但是查看代码是迄今为止最为有效的发现大多数错误的简便方法。而为了更好地查看代码，就要求开发人员遵循一定的代码书写规范，保证代码的可读性。这样也促进了开发水平的提升和潜在错误的减少。

目前，多数集成开发环境如 Eclipse 等都提供了 Coding Style(代码风格)功能，能自动提供代码缩进、代码块管理等功能，也能很方便地高亮显示不同类型的关键字、函数名、变量名等，这些功能都为代码审核提供了便利。为了更好地进行代码审核和减少错误的发生，还需要在书写代码时牢记以下几个原则：第一，代码一定要短小而清晰，能够用简单的算法实现的程序就不要使用复杂的算法，同时又要保证程序的清晰，减少使用复杂的条件语句等；第二，要使代码具备自解释性，采用能够直观地代表相应含义的变量名和方法名。

代码审核工作也可以采用所谓的同行评审的方式进行，即找一个具备类似开发技能的开发人员来评审他人的代码。



⊙ 利用输出调试

在计算机软件开发早期，没有任何调试工具，利用系统输出是最常用的一种调试方法。在集成开发工具不断发展的今天，很多初学者很少采用这种方式，甚至将其当作一个过时的调试方式。但其实它的简单性却正是它的威力所在。在怀疑可能出错的位置加入一句简单的System.out.println()语句，直接运行一次代码即可分析出错误所在，比起使用庞大的调试工具，设置断点一行行的执行代码要方便快捷得多。

由于 System.out.println()或者 JSP 中的 out.println()方法都是系统核心的组成部分，所以采用这种方法并不需要安装任何附加的程序和功能。另外，加入输出语句对程序的正常执行流程也不会有太多的影响，相反，调试工具的单步执行却可能有着极大的影响。例如在开发一些实时应用或者多线程应用时，采用调试器可能因为运行时序的变化而导致很难调试。

⊙ 日志记录

在前面就已经介绍过，Tomcat 会自动在日志中记录错误信息。事实上，所有系统捕获的异常和错误信息，Tomcat 服务器都会记录在 logs 目录下相应的日志文件中。因此，在发生错误时，仔细检查 logs 目录下的目录文件以获取更多的信息是一件非常重要的事。

同时，由于 Servlet 基础类提供了 log 方法，也可以在程序中将一些重要信息实时记录在日志文件中。这样，在应用程序运行时，阅读日志文件就可以得到之前真实的运行图，能起到多数调试器所能起到的作用而不用停止应用来进行调试。

⊙ 善用注释

前面提到的两种方法都是在源代码中加入一些调试信息来试图找出错误所在。除了这种做加法的方式以外，还可以做减法，即采用注释的方式。将怀疑具有错误嫌疑的部分代码采用注释的方式从运行代码中删除掉，检查剩下的代码是否正常运行，从而逐步缩小嫌疑目标，最终找出错误原因。

⊙ 使用调试程序

虽然还存在很多问题，但调试程序还是给开发调试工作带来了很多的帮助。不管是低级的 jdb 工具，还是高级的集成开发调试工具 Eclipse、JBuilder 等，都提供了设置断点、查看代码、单步执行以及检测和设置运行时变量等强大功能。作为开发人员需要做的是如何最大程度地利用好这些工具，而不是依赖甚至成为工具的奴隶。

14.2.5　异常处理

异常是程序执行时遇到的任何错误情况或意外行为。以下情况都可以引发异常：代码或调用的代码(如共享库)中有错误，操作系统资源不可用，虚拟机遇到意外情况(如无法验证代码)等。在 Java 环境中，异常是从 Exception 类继承的对象。异常从发生问题的代码区域引发，然

后沿堆栈向上传递，直到应用程序处理它或程序终止。

在 Java 中，将可能引发异常的代码块放在 try 块中，而将处理异常的代码放在 catch 块中。catch 块是一系列以关键字 catch 开头的语句，语句后跟异常的类型和要执行的操作。异常发生时，执行将终止，并且将控制交给最近的异常处理程序。这通常意味着不执行希望总调用的代码行。有些资源清理的工作(如关闭文件)必须得到执行，即使有异常发生。为实现这一点，我们可以使用 finally 块。finally 块总是执行，不论是否有异常发生。

下面的例子通过不同的 catch 语句对不同类型的异常进行处理。本例只是简单地打印所有的错误信息，读者可以很方便地将其改为向用户显示不同的中文错误信息。最终 finally 语句将强制执行 con.close 方法以释放连接。

```
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con=DriverManager.getConnection("jdbc:odbc:local");
    Statement stmt=con.createStatement();
    String queryStr="SELECT * FROM dbo.authors";
    ResultSet rs=stmt.executeQuery(queryStr);
    while(rs.next()) {
      System.out.println(rs.getString("au_fname"));
    }
}
catch (ClassNotFoundException ex) {
  ex.printStackTrace();
}
catch (SQLException sqle) {
  sqle.printStackTrace();
}
catch (Exception e) {
  e.printStackTrace();
}
finally {

   con.close();

}
```

14.3 上机练习

本章上机实验主要练习如何配置 JSP Web 应用程序，包括如何通过修改 conf/server.xml、conf/web.xml 以及 WEB-INF/web.xml 文件进行 JSP Web 应用程序的配置，以及如何进行错误调



试和异常处理等操作。

下面以使用 Eclipse 和 Tomcat 进行 JSP Web 应用程序 testTomcat 的部署为例来介绍。

(1) 使用资源管理器打开 Eclipse 所在文件夹，双击 Eclipse.exe 图标，打开 Eclipse 窗口。

(2) 确保 testTomcat 项目已经打开。选择【Project】|【Properties】命令打开 Properties for testTomcat 对话框设置项目属性。

(3) 在 Properties for testTomcat 对话框中单击展开【Tomcat】选项。然后选择【Export to WAR settings】选项卡。

(4) 在 WAR file to export 文本框中输入要导出的 WAR 文件名，如 C:\test.war。

(5) 单击【OK】按钮完成项目属性的设置。

(6) 在 Package Explorer 窗口中右击 testTomcat 项目。

(7) 选择【Tomcat project】|【Export to the WAR file sets in project properties】命令导出 WAR 文件。

(8) 操作成功完成后，会弹出 Operation successfully 对话框。单击【OK】按钮关闭对话框。

(9) 在步骤(4)中指定的目录下找到相应的 WAR 文件，将其拷贝至需要部署的服务器上 Tomcat 安装路径的 webapps 子目录下。

(10) 右键单击任务栏的 Tomcat 图标，选择【stop service】命令停止 Tomcat 服务。

(11) Tomcat 服务顺利停止之后，再选择【start service】命令重新启动 Tomcat 服务。

(12) 打开 Tomcat 安装路径 webapps 子目录，检查是否有对应 WAR 文件名的目录被创建(如 test)，并检查该目录下的内容是否与 testTomcat 项目的内容一致。

(13) 打开 IE 浏览器，输入相应的 URL 地址(如 http://www.deploysite.com/test)，检查 JSP Web 应用程序是否成功部署。



14.4 习题

14.4.1 填空题

1. JSP Web 应用程序采用__________作为配置系统的文件格式。

2. JSP 的主要配置文件分为机器级别的______________文件、______________文件和在应用程序级别的______________文件。

14.4.2 选择题

1. 以下配置选项中，一般情况下只用在 WEB-INF/web.xml 中的有(　　　　)。

A. Servlet　　B. Servlet-mapping　　C. context-param　　D. mime-mapping

2. 以下错误类型中，哪种错误类型是最难以预料和检查出来的(　　　)。

A. 配置错误　　B. 编译错误　　C. 运行时错误

14.4.3 问答题

1. 简述 JSP 配置系统的优点。
2. 简要列举常用的 JSP 调试方法和技巧。



第15章 JSP网站的构建实例

学习目标

教务网站是为了便于教务处发布和管理信息而开发的，旨在有效的帮助学校教务处及时的发布相关新闻，上传管理文件供师生下载，从而提高全校师生的工作效率。本章将详细介绍某大学教务网站的构建过程，该网站页面采用 JSP 技术生成，Web Server 则采用整合了 Apache 的 Tomcat 应用服务器，后台数据库为 Oralce 9i，客户端只需要一个 web 浏览器即可访问该教务网站。

本章重点

- 数据库设计与连接类
- 核心 JavaBean 类
- 性能测试

15.1 总体设计

15.1.1 系统架构

在设计 Web 教务系统时，考虑到该网站的运行环境为 Sun 公司的 Solaris 操作系统平台，于是采用基于 Java 语言的可移植 JSP 技术是较佳的选择。另外，为了提高代码的质量和复用度，决定采用基于 MVC 模式的 3 层 B/S 架构。

如图 15-1 所示，描述了该 web 教务系统的总体架构。

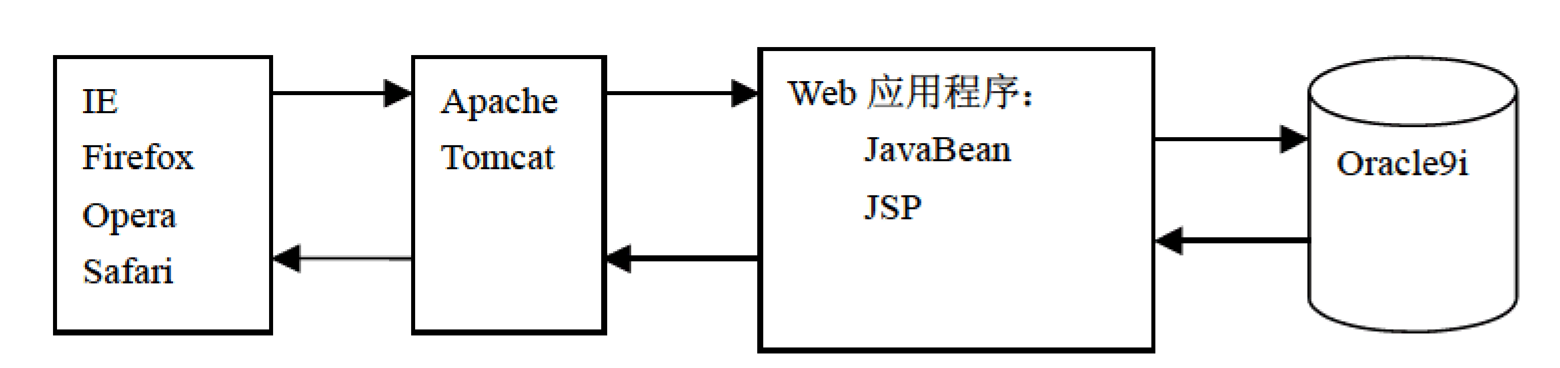


图 15-1　Web 教务系统的架构

15.1.2　Web 应用程序设计思路

为了构建出高效的 Web 系统，在应用程序的实现过程中，应尽可能采用先进的设计思路。

(1) 将内容的生成和显示分离

用 JSP 技术，Web 页面开发人员可以使用 HTML 或者 XML 标识来设计和格式化最终页面，并使用 JSP 标识或者小脚本来生成页面上的动态内容(内容是根据请求变化的，例如请求账户信息或者特定的一瓶酒的价格等)。生成内容的逻辑被封装在标识和 JavaBeans 组件中，并且捆绑在脚本中，所有的脚本在服务器端运行。由于核心逻辑被封装在标识和 JavaBeans 中，所以 Web 管理人员和页面设计者，能够编辑和使用 JSP 页面，而不影响内容的生成。在服务器端，JSP 引擎解释 JSP 标识和脚本，生成所请求的内容(例如，通过访问 JavaBeans 组件，使用 JDBC 技术访问数据库或者包含文件)，并且将结果以 HTML(或者 XML)页面的形式发送回浏览器。这既有助于作者保护自己的代码，又能保证任何基于 HTML 的 Web 浏览器的完全可用性。

(2) 利用可重用组件

绝大多数 JSP 页面依赖于可重用的、跨平台的组件(JavaBeans 或者 Enterprise JavaBeans 组件)来执行应用程序所要求的复杂的处理。开发人员能够共享和交换执行普通操作的组件，或者使得这些组件为更多的使用者和客户团体所使用。基于组件的方法加速了总体开发过程，并且使得各种组织在他们现有的技能和优化结果的开发努力中得到平衡。

(3) 采用表示简化开发

由于不是所有的 Web 页面开发人员都熟悉脚本语言，所以 Java Server Pages 技术封装了许多功能，这些功能是在与 JSP 相关的 XML 标识中生成动态内容所需要的。标准的 JSP 标识能够访问和实例化 JavaBeans 组件、设置或者检索组件属性、下载 Applet，以及执行用其他方法难于编码且耗时的功能。通过开发和定制标识库，可以扩展 JSP 技术。所以，第三方开发人员和其他人员可以为常用功能创建自己的标识库，这也使得 Web 页面开发得以简化。

15.1.3　设计模式的应用

在设计后台类代码时，不仅使用了面向对象设计技术，同时也使用了一些基本的设计模式。使用设计模式是为了可重用代码、让代码更容易被他人理解、同时保证代码的可靠性。系统在



实现 DBConnectionManager 这个类时候就采用了单身(Singleton)模式。单身模式是指一个类有且仅有一个实例，并且提供了一个全局的访问点，要实现单身模式，DBConnectionManager 类中必须有一些单身类的实例，且应该是静态的，如下所示：

```
static private DBConnectionManager instance;      //singleton
```

在初始化该 DBConnectionManager 类实例的时候，如果 instance 实例为空，则新建一个实例并返回，如果不为空，这说明该实例已经存在了，那么就不再创建新的实例，而返回原来的那个实例，代码如下所示：

```
static synchronized public DBConnectionManager getInstance()
{   if (instance == null)
{    instance = new DBConnectionManager();   }
    clients++;
     return instance;
 }
```

仅有上面的代码还不行，要实现 Singleton 模式，还必须将该类的构造方法设为私有的，代码如下：

```
private DBConnectionManager()
{ }
```

这样就可以禁止通过构造函数来实例化 DBConnectionManager 类。



15.2　数据库准备

Oracle9i 有 3 张安装光盘。点击第 1 张光盘的 setup.exe 将启动安装。选择安装位置，并设立 SID，其他都选择默认即可。安装后 Oracle 会提示用户输入特权用户 SYS 和 SYSTEM 的秘密。经过 Oracle 自动的配置后 Oracle 就安装完成了。安装以后要为 Oracle 建立一个 WEB 应用程序的用户：假定为 root，密码为 123。在这个用户下面建立 WEB 应用程序所需的数据库表。

在设计应用程序的后台数据库时，要充分考虑前台应用程序结构，根据 Web 应用程序的结构以及需要来设计后台数据库结构。教务处网站的主要作用是为了发布和更新学校或者教务处的新闻公告和文件等。为了这个达到这个目的，需要让教务处负责老师能够方便在后台就能够发布和管理这些新闻和文件。所以从这个角度出发，对数据库做了如下设计，一共引入 6 张主要的数据库表。下面逐一介绍。

- MASTER 表：保存管理员信息。为了提高系统安全性，Web 站点一般根据用户的权限划分不同管理等级，不同的等级对应不同的权限。
- CLASS 表：保存发布信息的不同类型。
- PICTURE 表：保存不同文头图片信息。

- NEWS 表：存储主页面上发布的通知公告和动态新闻。
- WJXZ 表：存储文件下载页面中的发布信息。
- GLWJ 表：存储管理文件页面中的发布信息。

15.2.1 MASTER 数据表

Master 表记录了网站管理人员的一些信息，如管理人员的用户名，口令，登录 IP，加入时间已经登录次数等等，Master 表同时也实现了对管理人员的分类管理。例如，某些权限的用户可以对网站的新闻信息管理，但是他却不能管理网站的其他管理人员。只有用户名为 root 的用户才拥有最高权限，可以管理网站的任何资源。Master 表的结构如表 15-1 所示。

表 15-1　Master 表

Name	Data type	Size	Scale	Nulls?	Default Value
ID	NUMBER	4	0	No	
NAME	VARCHAR2	20		No	
PASS	VARCHAR2	16		No	
IP	VARCHAR2	16		No	
TOTAL	NUMBER	10	0	No	
JOINDATE	VARCHAR2	10		No	
CLASSID	NUMBER	4	0	No	
LOGINNUM	NUMBER	10	0	No	

ID 字段是该管理人员的唯一标识符。Name 就是该管理人员的用户名。PASS 字段存储了该管理人员的口令密码。IP 是该管理人员的登录 IP。TOTAL 记录了该管理人员总共发布的信息的条数。JOINDATE 是管理人员的加入时间。CLASSID 对管理人员权限分类，CLASSID 目前只有 2 种取值，即 1 和 2，2 表示该管理人员为超级用户，他可以管理任何资源，甚至可以删除其他管理人员，1 表示该管理人员只能管理一般信息，比如添加信息，下载文件管理等等。LOGINNUM 表示该管理人员的登录次数。

15.2.2 CLASS 数据表

教务处网站上发布的信息一般有 8 种类型：它们分别是：通知，公告，国家文件，江苏省文件，学校文件，教改信息，教学成果，文件下载。在教务处网站首页会显示前 2 项信息：即教务处最新发布的 10 条通知公告，以及教务处最新发布的 10 条新闻，它们分别对应于数据库中该表的通知和公告 2 种类别。将这种对教务处文件的分类建立一张表，存储在数据库中。可



以对每一种文件类别建立一个 ID 号，并存储该类文件相对应的一些信息，比如类别名称，管理该类别文件的权限人等等。这样做的好处是：便于系统今后的维护和更新。例如：现在有 8 种文件类别，如果一段时间以后，教务处负责老师认为需要添加一种新的文件类别，叫做学生提议，那么只要将学生提议这种类别作为第 9 种文件类别插入到该数据库表中就可以了。而数据库的其他表都无需改动。将该表命名为：CLASS 表，意即文件类别。它的表结构如表 15-2 所示。

表 15-2　CLASS 表

Name	Data type	Size	Scale	Nulls?	Default Value
ID	NUMBER	4	0	No	
NAME	VARCHAR2	20		No	
MASTER	NUMBER	4	0	Yes	
PICAREA	NUMBER	2	0	No	

字段 ID 表示文件类别的唯一标识号，它是该表的主键。Name 字段表示文件类别名称。Master 表示管理该类别文件权限人。Picarea 是指明该类文件所使用的文件图片，存储在 PICTURE 表中。下面就会讲到。



15.2.3　PICTURE 数据表

Picture 表很简单，它的作用和 CLASS 表的作用是类似的。Picture 表存储了显示新闻时所采用的 2 种图片。一种图片是教务处通知文头纸格式图片，网站在发布教务处信息的时候一般采用这种图片，另一种图片是学校文头纸格式图片，网站在发布一些学校的信息的时候就采用这种图片。PICTURE 表的结构如表 15-3 所示。

表 15-3　PICTURE 表

Name	Data type	Size	Scale	Nulls?	Default Value
ID	NUMBER	5	0	No	
NARRATE	VARCHAR2	200		No	
ADDR	VARCHAR2	255		No	

15.2.4　NEWS 数据表

新闻表 NEWS 是整个数据库中最重要的一张表。它存储了所有教务处发布的新闻以及通知公告。它的表结构如表 15-4 所示。

表 15-4　NEWS 表

Name	Data type	Size	Scale	Nulls?	Default Value
ID	NUMBER	10	0	No	
TOPIC	VARCHAR2	255		No	
BODY	VARCHAR2	4000		No	
HITS	NUMBER	10	0	No	
ADDDATE	VARCHAR2	30		No	
ADDUSER	VARCHAR2	20		No	
ROOTID	NUMBER	4	0	No	
MARK	VARCHAR2	50		Yes	
PIC	NUMBER	5	0	Yes	
ATTACH	VARCHAR2	50		Yes	

字段 ID 表示该新闻的全局唯一标识符。TOPIC 表示一个新闻的标题。BODY 字段表示新闻的内容，由于新闻的正文可能比较长，所以给该字段用 VARCHAR2 类型分配了 4000 个字节。HITS 表示该新闻的点击率，这个可以用作教务处后台统计之用，便于教务处老师了解哪些新闻点击率比较高。ADDDATE 字段表示该新闻的添加时间。ADDUSER 代表了哪个用户添加了该条新闻。ROOTID 字段是指该新闻的类别是什么，也就是上面介绍的 CLASS 表中一种，这个字段只存储了一个类别 ID 号，通过 CLASS 表和 NEWS 表连接就可以知道该新闻类别的具体名称。MARK 字段表示该新闻的文号。PIC 字段用于表示在 HTML 网页上显示该新闻的时候，所使用的文件图片，目前一共有 2 种文件图片，它们存储在 PICTURE 表中。以后如果需要添加新的文件图片只要向 PICTURE 表中添加即可。ATTACH 字段存储了该新闻所带的附件。一般为一个 OFFICE 文档或一个图片等。

15.2.5　WJXZ 数据表

文件下载表记录了所有管理人员所发布的文件信息。文件下载表的结构如表 15-5 所示。

表 15-5　WJXZ 表

Name	Datatype	Size	Scale	Nulls?	Default Value
ID	NUMBER	5	0	No	
NAME	VARCHAR2	300		Yes	
TYPE1	CHAR	1		Yes	
TYPE2	CHAR	1		Yes	
HIDE	CHAR	1		Yes	
ATTACH	VARCHAR2	200		Yes	



ID 表示该文件的一个编号，Name 即为该文件的文件名，ATTACH 表示该文件所带的附件。

15.2.6 GLWJ 数据表

这个表记录了所有教务处发布的管理文件信息。教务处负责老师可以通过后台管理页面将新增的管理文件上传到服务器上面，然后供学校师生下载使用。该表的结构如表 15-6 所示。

表 15-6 GLWJ 表

Name	Data type	Size	Scale	Nulls?	Default Value
ID	NUMBER	3	0	No	
NAME	VARCHAR2	200		Yes	
TYPE	NUMBER	2	0	Yes	
HIDE	CHAR	1		Yes	
ATTACH	VARCHAR2	200		Yes	



到这里教务处网站的后台数据库就设计好了。下面如何使用 JSP 连接到 Oracle 数据库呢?

15.2.7 数据库连接类

要将 Web 应用程序与后台 Oracle 数据库相连，就要先将 OraHome\jdbc\lib 目录下的 classes12.jar 拷贝一份到 Tomcat 5.5\common\lib 下。并且需要添加环境变量 classpath=C:\Program Files\Apache Software Foundation\Tomcat 5.5\common\lib\classes12.jar。这里 OraHome 是 Oracle 安装目录。Classes12.jar 文件实际上就是 Oracle 厂商专门提供的用于连接 Oracle 数据库的 JDBC API。这样教务处网站就可以访问后台的 Oracle 数据库并向其中添加和管理数据了。此外，为了方便对于后台数据库的访问，采用面向对象程序设计思想实现了两个用于管理数据库连接的类：DBConnectionManager 和 DBConnect，其代码如下所示：

```
/*  DBConnectionManager */
package news.database;
import java.io.*;
import java.sql.*;
import java.util.*;
import java.util.Date;
public class DBConnectionManager {
  static private DBConnectionManager instance; // 唯一实例
  static private int clients;
```

```
private Vector drivers = new Vector();
private PrintWriter log;
private Hashtable pools = new Hashtable();
/**
 * 返回唯一实例。如果是第一次调用此方法，则创建实例
 *
 * @return DBConnectionManager 唯一实例
 */
static synchronized public DBConnectionManager getInstance() {
    if (instance == null) {
        instance = new DBConnectionManager();
    }
    clients++;
    return instance;
}
/**
 * 建构函数私有以防止其他对象创建本类实例
 */
private DBConnectionManager() {
    init();
}
/**
 * 将连接对象返回给由名字指定的连接池
 *
 * @param name 在属性文件中定义的连接池名字
 * @param con 连接对象
 */
public void freeConnection(String name, Connection con) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        pool.freeConnection(con);
    }
}
/**
 * 获得一个可用的(空闲的)连接。如果没有可用连接，且已有连接数小于最大连接数
 * 限制，则创建并返回新连接
 *
 * @param name 在属性文件中定义的连接池名字
 * @return Connection 可用连接或 null
 */
```

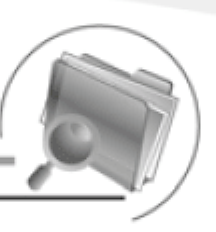
```
public Connection getConnection(String name) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        return pool.getConnection();
    }
    return null;
}
/**
 * 获得一个可用连接。若没有可用连接，且已有连接数小于最大连接数限制，
 * 则创建并返回新连接。否则，在指定的时间内等待其他线程释放连接。
 *
 * @param name  连接池名字
 * @param time  以毫秒计的等待时间
 * @return Connection  可用连接或 null
 */
public Connection getConnection(String name, long time) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        return pool.getConnection(time);
    }
    return null;
}
/**
 * 关闭所有连接，撤销驱动程序的注册
 */
public synchronized void release() {
    // 等待直到最后一个客户程序调用
    if (--clients != 0) {
        return;
    }
    Enumeration allPools = pools.elements();
    while (allPools.hasMoreElements()) {
        DBConnectionPool pool = (DBConnectionPool) allPools.nextElement();
        pool.release();
    }
    Enumeration allDrivers = drivers.elements();
    while (allDrivers.hasMoreElements()) {
        Driver driver = (Driver) allDrivers.nextElement();
        try {
            DriverManager.deregisterDriver(driver);
```

```
                log("撤销 JDBC 驱动程序  " + driver.getClass().getName()+"的注册");
            }
            catch (SQLException e) {
                log(e, "无法撤销下列 JDBC 驱动程序的注册: " + driver.getClass().getName());
            }
        }
    }
    /**
     * 根据指定属性创建连接池实例。
     *
     * @param props  连接池属性
     */
    private void createPools(Properties props) {
        Enumeration propNames = props.propertyNames();
        while (propNames.hasMoreElements()) {
        String name = (String) propNames.nextElement();
        if (name.endsWith(".url")) {
            String poolName = name.substring(0, name.lastIndexOf("."));
            String url = props.getProperty(poolName + ".url");
            if (url == null) {
                log("没有为连接池" + poolName + "指定 URL");
                continue;
            }
        String user = props.getProperty(poolName + ".user");
        String password = props.getProperty(poolName + ".password");
        String maxconn = props.getProperty(poolName + ".maxconn", "0");
        int max;
        try {
        max = Integer.valueOf(maxconn).intValue();
        }
        catch (NumberFormatException e) {
            log("错误的最大连接数限制: " + maxconn + " .连接池: " + poolName);
            max = 0;
        }
        DBConnectionPool pool = new DBConnectionPool(poolName, url, user, password, max);
        pools.put(poolName, pool);
        log("成功创建连接池" + poolName);
      }
    }
    }
```

```
/**
 * 读取属性完成初始化
 */
 private void init() {
 InputStream is = getClass().getResourceAsStream("/news.txt");
 Properties dbProps = new Properties();
 try {
   dbProps.load(is);
 }
 catch (Exception e) {
   System.err.println("不能读取属性文件。" +
  "请确保 db.properties 在 CLASSPATH 指定的路径中");
   return;
 }
 String logFile = dbProps.getProperty("logfile", "newslog.txt");
 try {
   log = new PrintWriter(new FileWriter(logFile, true), true);
 }
 catch (IOException e) {
   System.err.println("无法打开日志文件: " + logFile);
   log = new PrintWriter(System.err);
 }
 loadDrivers(dbProps);
 createPools(dbProps);
}
/**
 * 装载和注册所有 JDBC 驱动程序
 *
 * @param props  属性
 */
private void loadDrivers(Properties props) {
  String driverClasses = props.getProperty("driver");
  StringTokenizer st = new StringTokenizer(driverClasses);
  while (st.hasMoreElements()) {
  String driverClassName = st.nextToken().trim();
  try {
    Driver driver = (Driver)
    Class.forName(driverClassName).newInstance();
    DriverManager.registerDriver(driver);
    drivers.addElement(driver);
```



```
        log("成功注册 JDBC 驱动程序" + driverClassName);
      }
      catch (Exception e) {
        log("无法注册 JDBC 驱动程序: " +
        driverClassName + ", 错误: " + e);
      }
    }
  }
  /**
   * 将文本信息写入日志文件
   */
  private void log(String msg) {
//     log.println(new Date() + ": " + msg);
  }
  /**
   * 将文本信息与异常写入日志文件
   */
  private void log(Throwable e, String msg) {
    // log.println(new Date() + ": " + msg);
    // e.printStackTrace(log);
  }
  /**
   * 此内部类定义了一个连接池。它能够根据要求创建新连接，直到预定的最
   * 大连接数为止。在返回连接给客户程序之前，它能够验证连接的有效性。
   */
  class DBConnectionPool {
    private int checkedOut;
    private Vector freeConnections = new Vector();
    private int maxConn;
    private String name;
    private String password;
    private String URL;
    private String user;
    private String databaseUrl;
  /**
   * 创建新的连接池
   *
   * @param name  连接池名字
   * @param URL  数据库的 JDBC URL
   * @param user  数据库帐号，或 null
```



```
 * @param password 密码，或 null
 * @param maxConn 此连接池允许建立的最大连接数
 */
public DBConnectionPool(String name, String URL, String user, String password, int maxConn) {
  this.name = name;
  this.URL = URL;
  this.user = user;
  this.password = password;
  this.maxConn = maxConn;
  }
 /**
  * 将不再使用的连接返回给连接池
  *
  * @param con 客户程序释放的连接
  */
public synchronized void freeConnection(Connection con) {
  // 将指定连接加入到向量末尾
  freeConnections.addElement(con);
  checkedOut--;
  notifyAll();
}
/**
 * 从连接池获得一个可用连接。如没有空闲的连接且当前连接数小于最大连接
 * 数限制，则创建新连接。如原来登记为可用的连接不再有效，则从向量删除之，
 * 然后递归调用自己以尝试新的可用连接.
 */
public synchronized Connection getConnection() {
  Connection con = null;
  if (freeConnections.size() > 0) {
    // 获取向量中第一个可用连接
    con = (Connection) freeConnections.firstElement();
    freeConnections.removeElementAt(0);
    try {
      if (con.isClosed()) {
        log("从连接池" + name+"删除一个无效连接");
        // 递归调用自己，尝试再次获取可用连接
        con = getConnection();
      }
    }
    catch (SQLException e) {
```



```
            log("从连接池" + name+"删除一个无效连接");
            // 递归调用自己，尝试再次获取可用连接
            con = getConnection();
          }
        }
        else if (maxConn == 0 || checkedOut < maxConn) {
con = newConnection();
}
if (con != null) {
checkedOut++;
}
return con;
}
/**
* 从连接池获取可用连接。可以指定客户程序能够等待的最长时间
* 参见前一个 getConnection()方法。
*
* @param timeout  以毫秒计的等待时间限制
*/
public synchronized Connection getConnection(long timeout) {
long startTime = new Date().getTime();
Connection con;
while ((con = getConnection()) == null) {
try {
wait(timeout);
}
catch (InterruptedException e) {}
if ((new Date().getTime() - startTime) >= timeout) {
            // wait()返回的原因是超时
            return null;
          }
        }
        return con;
      }
 /**
  * 关闭所有连接
  */
 public synchronized void release() {
   Enumeration allConnections = freeConnections.elements();
   while (allConnections.hasMoreElements()) {
```

```
        Connection con = (Connection) allConnections.nextElement();
        try {
          con.close();
          log("关闭连接池" + name+"中的一个连接");
        }
        catch (SQLException e) {
          log(e, "无法关闭连接池" + name+"中的连接");
        }
      }
      freeConnections.removeAllElements();
    }
    /**
     * 创建新的连接
     */
    private Connection newConnection() {
    Connection con = null;
    try {
    if (user == null) {
      con = DriverManager.getConnection(URL);
    }
    else {
      con = DriverManager.getConnection(URL, user, password);
    }
      log("连接池" + name+"创建一个新的连接");
    }
    catch (SQLException e) {
      log(e, "无法创建下列 URL 的连接: " + URL);
      return null;
    }
      return con;
  }
  }
  }

/* 如下为 DBConnect 类 */
package news.database;
import java.sql.*;
import news.database.DBConnectionManager;
public class DBConnect {
  private Connection conn = null;
```



```
private Statement stmt = null;
private PreparedStatement prepstmt = null;
private DBConnectionManager dcm=null;
    void init() {
    dcm = DBConnectionManager.getInstance();
  conn = dcm.getConnection("mysql");
    }
/*  构造数据库的连接和访问类  */
public DBConnect() throws Exception {
          init();
  stmt = conn.createStatement();
}
    public DBConnect(int resultSetType, int resultSetConcurrency)
                throws Exception {
          init();
          stmt = conn.createStatement(resultSetType, resultSetConcurrency);
    }
    /*  构造数据库的连接和访问类
     *  预编译 SQL 语句
     * @param sql SQL 语句
     */
public DBConnect(String sql) throws Exception {
          init();
  this.prepareStatement(sql);
}
public DBConnect(String sql, int resultSetType, int resultSetConcurrency)
                throws Exception {
          init();
  this.prepareStatement(sql, resultSetType, resultSetConcurrency);
}
/*返回连接
  * @return Connection  连接
  */
public Connection getConnection() {
  return conn;
}
/* PreparedStatement
  * @return sql  预设 SQL 语句
  */
public void prepareStatement(String sql) throws SQLException {
```

```
    prepstmt = conn.prepareStatement(sql);
  }
  public void prepareStatement(String sql, int resultSetType, int resultSetConcurrency)
                    throws SQLException {
    prepstmt = conn.prepareStatement(sql, resultSetType, resultSetConcurrency);
  }
  /*  设置对应值
   * @param index  参数索引
   * @param value  对应值
   */
  public void setString(int index,String value) throws SQLException {
    prepstmt.setString(index, value);
  }
  public void setInt(int index,int value) throws SQLException {
    prepstmt.setInt(index,value);
  }
  public void setBoolean(int index,boolean value) throws SQLException {
    prepstmt.setBoolean(index,value);
  }
  public void setDate(int index,Date value) throws SQLException {
    prepstmt.setDate(index,value);
  }
  public void setLong(int index，long value) throws SQLException {
    prepstmt.setLong(index,value);
  }
  public void setFloat(int index，float value) throws SQLException {
    prepstmt.setFloat(index,value);
  }
  public void setBytes(int index，byte[] value) throws SQLException{
    prepstmt.setBytes(index,value);
  }
      public void clearParameters()
            throws SQLException
      {
            prepstmt.clearParameters();
            prepstmt=null;
      }
  /*  返回预设状态   */
  public PreparedStatement getPreparedStatement() {
    return prepstmt;
```

```
}
/* 返回状态
 * @return Statement  状态
 */
public Statement getStatement() {
  return stmt;
}
/* 执行 SQL 语句返回字段集
 * @param sql SQL 语句
 * @return ResultSet  字段集
 */
public ResultSet executeQuery(String sql) throws SQLException {
  if (stmt != null) {
    return stmt.executeQuery(sql);
  }
  else return null;
}
public ResultSet executeQuery() throws SQLException {
  if (prepstmt != null) {
    return prepstmt.executeQuery();
  }
  else return null;
}
/* 执行 SQL 语句
 * @param sql SQL 语句
 */
public void executeUpdate(String sql) throws SQLException {
  if (stmt != null)
    stmt.executeUpdate(sql);
}
public void executeUpdate() throws SQLException {
  if (prepstmt != null)
    prepstmt.executeUpdate();
}
/* 关闭连接 */
public void close() throws Exception {
  if (stmt != null)   {
    stmt.close();
    stmt = null;
  }
```



```
    if (prepstmt != null) {
      prepstmt.close();
      prepstmt = null;
    }
    if (conn!=null)
    {
      dcm.freeConnection("mysql",conn);
      }
  }
}
```

有了上述的 2 个数据库连接类，后面访问数据库就可以直接通过创建 DBConnect 类实例来完成。

15.3 核心 JavaBean



鉴于篇幅所限，这里只能选取系统中重要的某些 JavaBean 做一介绍，这些 JavaBean 都与前面介绍过的 NEWS 数据表有关。众所周知，新闻表 NEWS 是整个数据库中最重要的一张表，它用来存储所有教务处发布的新闻以及通知公告。为了能够往数据库中添加 news 数据、修改 news 数据以及删除 news 数据，首先应先设计一个 News 类：

```
/*News 类 */
package news.news;
import news.database.DBConnect;
import java.util.*;
public class News {
  public String topic,body,adddate,adduser,mark,attach;
  public int ID,hits,rootID,link,pic;
  public News(){};
  public int getID(){
    return ID;
  }
  public int getRootID(){
    return rootID;
  }
  public String getTopic(){
    return topic;
  }
  public String getBody(){
```

```
    return body;
}
public int getHits(){
    return hits;
}
public String getAdddate(){
    return adddate;
}
public String getAdduser(){
    return adduser;
}
public int getPic(){
    return pic;
}
public String getMark(){
    return mark;
}
public int getLink(){
    return link;
}
public String getAttach(){
    return attach;
}
//取新闻排列的 ID
public void setID(int i){
    this.ID = i;
}
//取新闻所属的类别 ID
public void setRootID(int i){
    this.rootID = i;
}
//取新闻标题
public void setTopic(String s){
    this.topic = s;
}
//取新闻内容
public void setBody(String s){
    this.body = s;
}
//取新闻点击次数
```





```
  public void setHits(int i){
    this.hits = i;
  }
  //取新闻加入时间
  public void setAdddate(String s){
    this.adddate = s;
  }
  //取新闻添加用户
  public void setAdduser(String s){
    this.adduser = s;
  }
  //取新闻图片
  public void setPic(int i){
    this.pic = i;
  }
  public void setMark(String s){
    this.mark = s;
  }
  public void setLink(int i){
    this.link = i;
  }
  public void setAttach(String s){
    this.attach = s;
  }
}
```

定义完 News 类，可以再设计一个 NewsControl 类。它直接从 News 类继承，当需要进行发布新闻或者更新新闻时，首先创建 NewsControl 类实例，然后通过调用基类 News 提供的 set 方法对其各个属性进行赋值，接着就可以调用 NewsControl 类提供的方法(该方法可以直接访问到具体 News 类实例的属性值)来进行数据的插入或者更新操作了。NewsControl 类的代码如下所示：

```
/*   NewsControl 类   */
package news.admin;
import news.database.DBConnect;
import news.news.*;
import news.admin.*;
import news.util.*;
import java.sql.*;
import java.util.*;
public class NewsControl extends News{
```



```
public ResultSet rs;
private int newsid;
public NewsControl(){};
/*
* 添加新闻
*/
public int insertnews(){
 DBConnect dbc = null;
 try{
   dbc = new DBConnect();
   dbc.prepareStatement("INSERT INTO news ( id,topic,body,adddate,adduser,rootid,pic,hits,mark,link,attach )
VALUES ( news_sequence.NEXTVAL,?,?,?,?,?,?,0,?,?,?)");
   dbc.setString(1,topic);
   dbc.setString(2,body);
   dbc.setString(3,GetDate.getStringDate());
   dbc.setString(4,adduser);
   dbc.setInt(5,rootID);
   dbc.setInt(6,pic);
   dbc.setString(7,mark);
   dbc.setInt(8,link);
   dbc.setString(9,attach);
   //dbc.setString(9,"999.bmp");
   dbc.executeUpdate();
   dbc.prepareStatement("SELECT max(id) FROM news");
   rs = dbc.executeQuery();
   if(rs.next())
       newsid = rs.getInt(1);
    }catch(Exception e){
    System.err.println(e);
   }finally{
    try{
      dbc.close();
    }catch(Exception e){
      e.printStackTrace();
    }
   }
 return newsid;
}

/*
* 修改新闻
*/
public void modifynews(){
```

```
    DBConnect dbc = null;
     try{
       dbc = new DBConnect();
       dbc.prepareStatement("UPDATE news SET topic=?,body=?,rootid=?,pic=?,mark=?,link=?, attach=? WHERE
id=?");
       dbc.setString(1,topic);
       dbc.setString(2,body);
       dbc.setInt(3,rootID);
       dbc.setInt(4,pic);
       dbc.setString(5,mark);
       dbc.setInt(6,link);
       dbc.setString(7,attach);
       dbc.setInt(8,ID);
       dbc.executeUpdate();
     }catch(Exception e){
       e.printStackTrace();
     }
     finally{
     try{
       dbc.close();
     }catch(Exception e){
       e.printStackTrace();
       }
     }
    }
    /*
    * 删除新闻
    */
    public void delnews(){
     try{
       DBConnect dbc = new DBConnect();
       dbc.prepareStatement("delete from news WHERE id=?");
       dbc.setInt(1,ID);
       dbc.executeUpdate();
       dbc.close();
     }catch(Exception e){
       e.printStackTrace();
     }
    }
    /*
    * 新闻浏览次数加一
    */
    public void addhits(){
```

```
DBConnect dbc = null;
 try{
   dbc    = new DBConnect();
   dbc.prepareStatement("UPDATE news SET hits = hits +1 WHERE id = ?");
   dbc.setInt(1,ID);
   dbc.executeUpdate();
 }catch(Exception e){
   System.err.println(e);
 }finally{
 try{
   dbc.close();
 }catch(Exception e){
   e.printStackTrace();
   }
 }
}
}
```

上述 NewsControl 类中提供了 insertnews()、modifynews()、delnews()以及 addhits()等方法来实现 news 数据记录的录入、修改、删除以及访问计数，因此，该类主要用于网站后台管理中。另外，在网站前台主页中需要让最近发布的一些新闻信息能够显示出来，这就又涉及到对数据库 NEWS 表的读访问，为此，又定义了如下 DispNews 类：

```
/* DispNews 类 */
package news.news;
import news.database.DBConnect;
import news.news.*;
import news.util.*;
import java.sql.*;
import java.util.*;
public class DispNews extends News{
public ResultSet rs;
public DispNews(){};
/*
* 根据 rootid 得到某栏目所有的新闻
*/
public Vector rootidToNews() {
 DBConnect dbc = null;
 Vector newsInfoVector = new Vector();
   try{
   dbc    = new DBConnect();
   dbc.prepareStatement("SELECT * FROM news WHERE rootid = ? order by id desc");
   dbc.setInt(1,rootID);
```

```
  rs = dbc.executeQuery();
    while(rs.next()){
    News news = new News();
    news.setID(rs.getInt("id"));
    news.setTopic(rs.getString("topic"));
    news.setBody(rs.getString("body"));
    news.setHits(rs.getInt("hits"));
    news.setAdddate(rs.getString("adddate"));
    news.setAdduser(rs.getString("adduser"));
    news.setRootID(rs.getInt("rootid"));
    news.setPic(rs.getInt("pic"));
    news.setMark(rs.getString("mark"));
    news.setLink(rs.getInt("link"));
    news.setAttach(rs.getString("attach"));
    newsInfoVector.add(news);
  }
 }catch(Exception e){
  System.err.println(e);
 }finally{
  try{
    dbc.close();
  }catch(Exception e){
    e.printStackTrace();
  }
 }
 return newsInfoVector;
}
/*
* 根据 ID 得到新闻
*/
public News idToNews() {
 DBConnect dbc = null;
 News news = new News();
 try{
  dbc   = new DBConnect();
  dbc.prepareStatement("SELECT * FROM news WHERE id = ?");
  dbc.setInt(1,ID);
  rs = dbc.executeQuery();
    if(rs.next()){
    news.setID(rs.getInt("id"));
    news.setTopic(rs.getString("topic"));
    news.setBody(rs.getString("body"));
    news.setHits(rs.getInt("hits"));
```



```
        news.setAdddate(rs.getString("adddate"));
        news.setAdduser(rs.getString("adduser"));
        news.setRootID(rs.getInt("rootid"));
        news.setPic(rs.getInt("pic"));
        news.setMark(rs.getString("mark"));
        news.setLink(rs.getInt("link"));
        news.setAttach(rs.getString("attach"));
      }
    }catch(Exception e){
      System.err.println(e);
    }finally{
      try{
        dbc.close();
      }catch(Exception e){
        e.printStackTrace();
      }
    }
    return news;
  }
  public Vector allNews() {
    DBConnect dbc = null;
    Vector allNewsVector = new Vector();
    try{
      dbc    = new DBConnect();
      dbc.prepareStatement("SELECT * FROM news order by hits desc");
      rs = dbc.executeQuery();
        while(rs.next()){
        News news = new News();
        news.setID(rs.getInt("id"));
        news.setTopic(rs.getString("topic"));
        news.setBody(rs.getString("body"));
        news.setHits(rs.getInt("hits"));
        news.setAdddate(rs.getString("adddate"));
        news.setAdduser(rs.getString("adduser"));
        news.setRootID(rs.getInt("rootid"));
        news.setPic(rs.getInt("pic"));
        news.setMark(rs.getString("mark"));
        news.setLink(rs.getInt("link"));
        news.setAttach(rs.getString("attach"));
        allNewsVector.add(news);
      }
    }catch(Exception e){
      System.err.println("error:"+e);
```

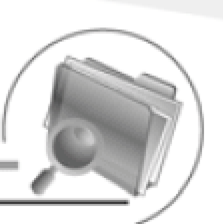


```
    }finally{
      try{
        dbc.close();
      }catch(Exception e){
        e.printStackTrace();
      }
    }
    return allNewsVector;
  }
  /*
  * 计算某类别新闻的总数
  */
  public int newsNum() {
    DBConnect dbc = null;
    int newsCount = 0;
    try{
      dbc    = new DBConnect();
      dbc.prepareStatement("SELECT count(*) FROM news WHERE rootid = ?");
      dbc.setInt(1,rootID);
      rs = dbc.executeQuery();
      if(rs.next())newsCount = rs.getInt(1);
    }catch(Exception e){
      System.err.println(e);
    }finally{
      try{
        dbc.close();
      }catch(Exception e){
        e.printStackTrace();
      }
    }
    return newsCount;
  }
  /*
  * 计算某新闻的最大点击数
  */
  public int maxHit() {
    DBConnect dbc = null;
    int maxhit = 0;
    try{
      dbc    = new DBConnect();
      dbc.prepareStatement("SELECT max(hits) FROM news");
      rs = dbc.executeQuery();
```



```
      if(rs.next())maxhit = rs.getInt(1);
    }catch(Exception e){
      System.err.println(e);
    }finally{
      try{
        dbc.close();
      }catch(Exception e){
        e.printStackTrace();
      }
    }
    return maxhit;
  }
  public String search;
  public String getSearch(){
    return search;
  }

  public void setSearch(String s){
    this.search = s;
  }
  /*
  * 根据条件在所有的新闻中查询
  */
  public Vector searchNews() {
    DBConnect dbc = null;
    Vector searchNewsInfoVector = new Vector();
    try{
      dbc    = new DBConnect();
      dbc.prepareStatement("SELECT * FROM news WHERE topic like ? order by id desc");
      //dbc.setBytes(1,search.getBytes("GB2312"));
      dbc.setString(1,("%"+search+"%"));
      rs = dbc.executeQuery();
        while(rs.next()){
        News news = new News();
        news.setID(rs.getInt("id"));
        news.setTopic(rs.getString("topic"));
        news.setBody(rs.getString("body"));
        news.setHits(rs.getInt("hits"));
        news.setAdddate(rs.getString("adddate"));
        news.setAdduser(rs.getString("adduser"));
        news.setRootID(rs.getInt("rootid"));
        news.setPic(rs.getInt("pic"));
```



```
      news.setMark(rs.getString("mark"));
      news.setLink(rs.getInt("link"));
      news.setAttach(rs.getString("attach"));
      searchNewsInfoVector.add(news);
    }
  }catch(Exception e){
    System.err.println(e);
  }finally{
    try{
      dbc.close();
    }catch(Exception e){
      e.printStackTrace();
    }
  }
  return searchNewsInfoVector;
 }
}
```

上述 DispNews 类提供了根据 rootid 得到某栏目所有的新闻、根据 ID 得到新闻、计算某类别新闻的总数、计算某新闻的最大点击数以及根据条件在所有的新闻中查询等功能，在前台主页中显示新闻时就可以根据需要有选择地来调用它们，这样就将处理事务逻辑的代码集中到 JavaBean 端来实现，避免了在 JSP 页面中过多地嵌入 Java 代码，既增加了可读性又提高了代码的复用性。下面，再来看一下网站中与 News 新闻相关的 JSP 页面。



15.4　网站页面

设计好了前述的 Java 类后，下面需要考虑的就是如何设计编写 JSP 页面。通常的做法是在做好页面布局和美工等的基础上，通过嵌入简短的 Java 代码块调用 JavaBean 的功能，实现数据库中表记录信息在后台 JSP 管理界面的录入、修改、删除、统计以及在前台 JSP 页面上的动态显示。为了控制篇幅，在介绍网站的 JSP 页面时，仅列出其中的一些关键 Java 代码，其他与页面布局、显示和美工等静态内容部分(不涉及数据库访问的)相关的则不予列出。

15.4.1　后台管理界面

管理员通过账号和密码可以登录后台界面，以对网站信息内容进行维护和管理。如图 15-2 所示为登录界面。

=> 管理员登录
[忘记密码] [返回]
用户名：
密　码：
登 陆

图 15-2　后台登录界面

登录成功后，即进入如图 15-3 所示的管理界面。

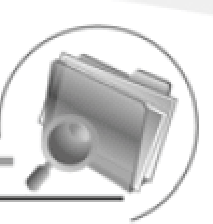


图 15-3　后台管理界面

从如图 15-3 所示的界面可以看出，后台管理能够进行(新闻)信息的添加(即发布)，信息的编辑和删除信息等操作，同时还可以进行管理员的管理、栏目管理、图片管理、校历管理、管理文件管理以及文件下载管理等操作。下面仅列出新闻信息管理的相关页面，如图 15-4 所示。

录入完上述信息内容后，单击发表按钮即可转入保存信息的 JSP 页面的执行，该保存信息的 JSP 页面中的关键 Java 代码如下所示：

```
<%try{
        String subject = new String(request.getParameter("subject").getBytes("ISO8859_1"),"GBK");
        String content = new String(request.getParameter("Content").getBytes("ISO8859_1"),"GBK");
        String themark = new String(request.getParameter("themark").getBytes("ISO8859_1"),"GBK");
        int pic = Integer.parseInt(request.getParameter("pic"));
```



```
            int link = Integer.parseInt(request.getParameter("link"));
            String attach = new String(request.getParameter("attach").getBytes("ISO8859_1"),"GBK");
            int rootid = Integer.parseInt(request.getParameter("classid"));
            String master = (String) session.getValue("userName_s");
            NewsControl newscontrol= new NewsControl();
            newscontrol.setTopic(subject);
            newscontrol.setBody(CodeFilter.toHtml(content));
            newscontrol.setAdddate(GetDate.getStringDate());
            newscontrol.setAdduser(master);
            newscontrol.setRootID(rootid);
            newscontrol.setPic(pic);
            newscontrol.setLink(link);
            newscontrol.setMark(themark);
            newscontrol.setAttach(attach);
            int newsid = newscontrol.insertnews();
        }
%>
```





图 15-4　后台信息发布页面

上述代码首先从 JSP 的 request 对象获取前面页面(即信息发布页面)中的输入值，然后创建 NewsControl 对象，接着对该对象的 News 子对象进行赋值，最后通过调用 insertnews()方法来实现该条新闻信息的录入。图 15-5 所示为新闻信息的编辑(即修改)页面。



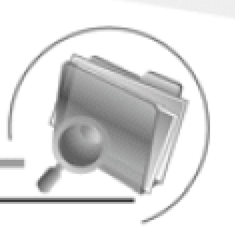
图 15-5　后台信息修改页面



新闻信息的修改与录入差不多，只是在保存修改后的信息时，调用的不是 insertnews()方法，而是 modifynews()方法。至于信息的删除操作，只要单击图 15-3 中处于右边的【删除信息】链接即可，该链接其实是一个 JSP 页面，它的关键 Java 代码如下：

```
<% //验证用户是否有权限
 int id = Integer.parseInt(request.getParameter("id"));
 DispNews dispnews = new DispNews();
 dispnews.setID(id);
 News news = dispnews.idToNews();
 DispMaster dispmaster = new DispMaster();
 dispmaster.setUserName(userName);
 Master master = dispmaster.nameToMaster();
 if(news.getRootID()!=master.getClassid() && !userName.equals("admin")){
     response.sendRedirect("error1.jsp"); }
 else{
//具备删除权限
     NewsControl newscontrol= new NewsControl();
     newscontrol.setID(id);      //设置即将被删除信息的 id 号
     newscontrol.delnews();   //调用能够删除信息的 delnews()方法
 }
%>
```

后台管理的 JSP 页面中，除了需要上述关键 Java 代码块外，同时也需要其他相关元素，如



当信息发布成功后、信息正确修改后以及信息被成功删除后等，页面都应有相应的反馈给用户，告诉用户操作是否成功。

15.4.2　前台首页

网站的首页如图 15-6 所示，它提供了教务新闻和教务动态的显示、其他前台页面的入口链接以及网站信息的搜索等。



图 15-6　网站前台首页



15.5　性能测试和优化

15.5.1　性能测试

采用知名的性能测试软件 Jmeter 来对实现后的 Web 教务系统做性能测试。在 Jmeter 中建立一个测试计划首先需要增加一个 Thread Group，它用来模拟用户向 Web 服务器发出的大量 Http 请求。Thread Group 有 3 个和负载信息相关的参数：Number of Threads 即发送请求的用户数目，Ramp-up period 每个请求发生的总时间间隔，Loop Count 请求发生的重复次数。然后还需要设定 Http 请求的参数，如设定 Protocol，默认为 Http，服务器 IP，Web 应用程序发布路径等，其他一些参数设置这里就不做说明了。

这里模拟了对其同时发出 500 个 Http 请求的测试，从图 15-7 可以看出，所有请求都是成功的，平均响应时间为 16ms，系统的吞吐量为 48.9/s。

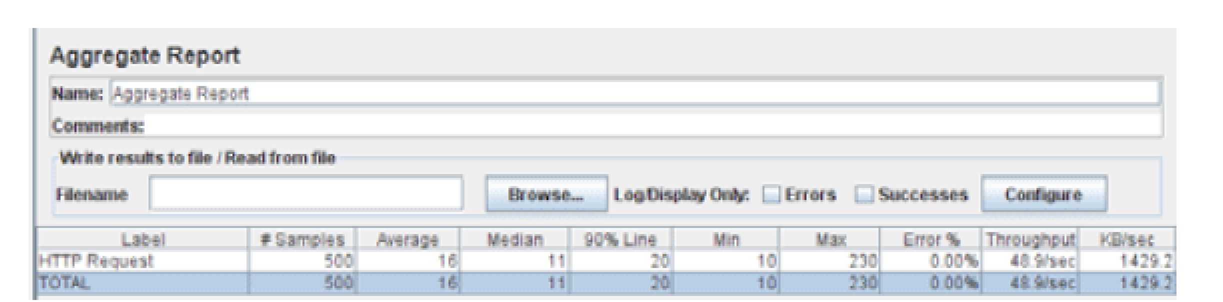


图 15-7 测试结果

15.5.2 系统优化

为了提高 Web 系统的性能，从软件角度作了以下优化措施：

(1) 服务器专门用于提供 Web 服务，对配置文件进行编辑以去掉任何其他的服务。通常，Web 服务器会运行 ftp、nfs、dhcp、dns 和其他非必需的守护进程。计算机所需共享的资源越少，那么它对其主要任务的响应能力越高。作为回报，安全性也更高，因为可供攻击的漏洞少了。



(2) 对应用服务器 Tomcat 进行优化配置，如系统上线后可以不必再作 JSP 文件是否已被修改的检查等。

(3) 采用数据库连接池技术。建立 Web 应用程序与 Oracle 之间的 TCP 连接是一项费时的操作，该技术能重用到数据库的连接，而不是每次请求都建立新的 TCP 连接。

如图 15-8 所示为优化后的系统测试结果。

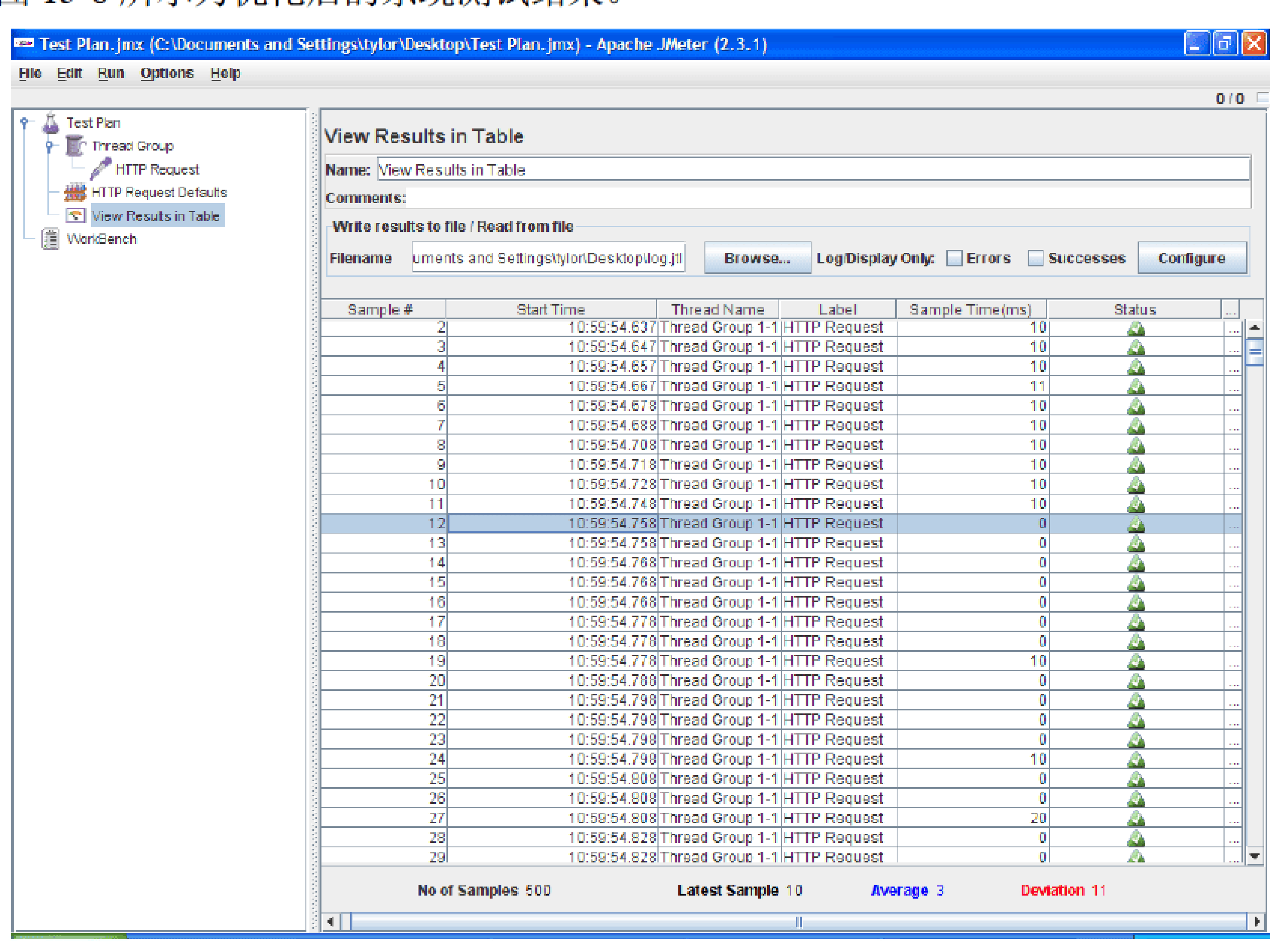


图 15-8 系统优化后的测试结果

从图 15-8 中可以看到：对 Web 系统做了相应优化以后，性能提升明显，平均响应时间从原来的 16ms 降低到目前的 3ms。